

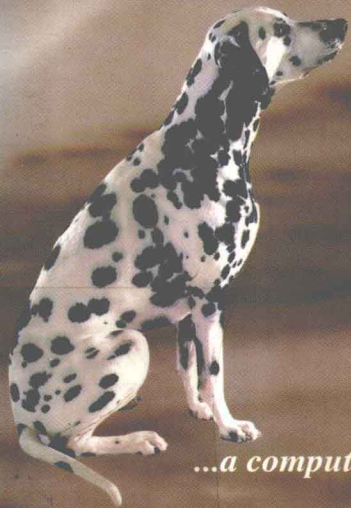
JAMSA'S

C/C++

PROGRAMMER'S BIBLE

The Ultimate Guide to
C/C++ Programming

By Kris Jamsa, Ph.D.
and Lars Klander



...a computer user's best friend®

**Companion
CD-ROM includes**
Borland Turbo C++ Lite—
Everything you need to
create C++ programs!

JAMSA'S

C/C++

PROGRAMMER'S BIBLE

The Ultimate Guide to
C/C++ Programming

By Kris Jamsa, Ph.D.
and Lars Klander



JAMSA'S C/C++ PROGRAMMER'S BIBLE

Published by

Jamsa Press
2975 S. Rainbow Blvd., Suite I
Las Vegas, NV 89102
U.S.A.

<http://www.jamsapress.com>

For information about the translation or distribution of any Jamsa Press book, please write to Jamsa Press at the address listed above.

Jamsa's C/C++ Programmer's Bible

Copyright © 1998 by Jamsa Press. All rights reserved. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any format or by any means, or stored in a database or retrieval system, without the prior written permission of Jamsa Press.

Printed in the United States of America.

987

ISBN 1-884133-25-8

Publisher

Debbie Jamsa

Content Manager

Todd Peterson

Composition

Eugene Marks
James Rehrauer
Nelson Yee

Proofers

Rosemary Pasco
Jeanne K. Smith

Technical Advisor

Phil Schmauder

Cover Photograph

O'Gara/Bissell

Illustrators

Eugene Marks
James Rehrauer
Nelson Yee

Indexer

John Bianchi

Director of Publishing Operations

Janet Lawrie

Cover Design

Marianne Helm
James Rehrauer

Copy Editors

Ann Edwards
Dorothy Oppenheimer
Renée Wesberry

Technical Editors

C.J. Bockmann
LingYan Tang

This book identifies product names and services known to be trademarks or registered trademarks of their respective companies. They are used throughout this book in an editorial fashion only. In addition, terms suspected of being trademarks or service marks have been appropriately capitalized. Jamsa Press cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

The information and material contained in this book are provided "as is," without warranty of any kind, express or implied, including, without limitation, any warranty concerning the accuracy, adequacy, or completeness of such information or material or the results to be obtained from using such information or material. Neither Jamsa Press nor the author shall be responsible for any claims attributable to errors, omissions, or other inaccuracies in the information or material contained in this book, and in no event shall Jamsa Press or the author be liable for direct, indirect, special, incidental, or consequential damages arising out of the use of such information or material.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in rendering professional service or endorsing particular products or services. If legal advice or other expert assistance is required, the services of a competent professional should be sought.

Jamsa Press is an imprint of Gulf Publishing Company:

Gulf Publishing Company
Book Division
P.O. Box 2608
Houston, TX 77252-2608
U.S.A.

<http://www.gulfpub.com>

Table of Contents

GETTING STARTED WITH C

An Introduction to Programming	1
Creating an ASCII Source File	2
Compiling Your C Program	3
Understanding Syntax Errors	4
The Structure of a Typical C Program	5
Adding Statements to Your Program	6
Displaying Output on a New Line	7
C Considers Upper- and Lowercase Letters as Different	8
Understanding Logic Errors (Bugs)	9
Understanding the Program Development Process	10
Understanding the File Types	11
Better Understanding the Linker	12
Understanding Header Files	13
Helping the Compiler Locate Header Files	14
Speeding Up Compilations	15
Commenting Your Programs	16
Improving Your Program Readability	17
Paying Attention to Compiler Warning Messages	18
Controlling Compiler Warnings	19
Using Comments to Exclude Program Statements	20
Understanding the Importance of Names	21
Understanding the Semicolon	22
Introducing Variables	23
Assigning a Value to a Variable	24
Understanding Variable Types	25
Declaring Multiple Variables of the Same Type	26
Commenting Your Variables at Declaration	27
Assigning Values to Variables at Declaration	28
Initializing Multiple Variables During Declaration	29
Using Meaningful Variable Names	30
Understanding C's Keywords	31
Understanding Variables of Type int	32
Understanding Variables of Type char	33
Understanding Variables of Type float	34
Understanding Variables of Type double	35
Assigning Values to Floating-Point Values	36
Understanding Type Modifiers	37
Understanding the unsigned Type Modifier	38
Understanding the long Type Modifier	39
Combining the unsigned and long Type Modifiers	40
Working with Large Values	41
Understanding the register Type Modifier	42
Understanding the short Type Modifier	43
Omitting int from Modified Declarations	44
Understanding the signed Type Modifier	45
Multiple Assignment Operations	46
Assigning One Variable Type's Value to a Different Type	47
Creating Your Own Types	48
Assigning a Hexadecimal or Octal Value	49
Understanding Overflow	50
Understanding Precision	51
Assigning Quotes and Other Characters	52
Getting Started with printf	53
Displaying Values of Type int Using printf	54
Printing an Octal or Hexadecimal Integer Value	55
Displaying Values of Type unsigned int Using printf	56
Displaying Values of Type long int Using printf	57
Displaying Values of Type float Using printf	58
Displaying Values of Type char Using printf	59
Displaying Floating-Point Values in an Exponential Format	60
Displaying Floating-Point Values	61
Displaying a Character String Using printf	62
Displaying a Pointer Address Using printf	63
Preceding a Value with a Plus or Minus Sign	64
Formatting an Integer Value Using printf	65
Zero-Padding Integer Output	66
Displaying a Prefix Before Octal and Hexadecimal Values	67
Formatting a Floating-Point Value Using printf	68
Formatting Exponential Output	69
Left-Justifying printf's Output	70
Combining printf Format Specifiers	71
Wrapping a Character String to the Next Line	72
Displaying Near and Far Strings	73
Working with printf's Escape Characters	74
Determining the Number of Characters printf has Displayed ...	75
Using printf's Return Value	76
Using the ANSI Device Driver	77
Using the ANSI Driver to Clear your Screen Display	78
Using the ANSI Driver to Display Screen Colors	79
Using the ANSI Driver to Position the Cursor	80
Performing Basic Math Operations in C	81
Understanding Modulo Arithmetic	82
Understanding Operator Precedence and Associativity	83
Forcing the Order of Operator Evaluation	84
Understanding C's increment Operator	85
Understanding C's decrement Operator	86
Understanding a Bitwise OR Operation	87
Understanding a Bitwise AND Operation	88
Understanding a bitwise exclusive OR Operation	89
Understanding a Bitwise inverse Operation	90
Applying an Operation to a Variable's Value	91
Understanding C's conditional Operator	92
Understanding C's sizeof Operator	93
Performing a Bitwise Shift	94
Performing a Bitwise Rotation	95
Understanding Conditional Operators	96
Understanding Iterative Processing	97
Understanding How C Represents True and False	98
Testing a Condition with if	99
Understanding Simple and Compound Statements	100
Testing for Equality	101
Performing Relational Tests	102
Performing a Logical AND to Test Two Conditions	103
Performing a Logical OR to Test Two Conditions	104
Performing a Logical NOT Operation	105
Assigning the Result of a Condition	106
Declaring Variables Within Compound Statements	107
Using Indentation to Improve Readability	108
Using Extended Ctrl+Break Checking	109
Testing Floating-Point Values	110
Looping Forever	111
Testing an Assignment	112
Beware of if-if-else Statements	113
Performing Statements a Specific Number of Times	114
Parts of the for Statement are Optional	115
Decrementing Values in a for Statement	116
Controlling the for Loop Increment	117
Using for Loops with char and float Values	118
Understanding a NULL Loop	119
Understanding an Infinite Loop	120
Using C's Comma Operator Within a for Loop	121
Avoid Changing the Control Variable's Value in a for Loop	122
Repeating One or More Statements Using a while Loop	123
Understanding the Parts of a while Loop	124
Repeating One or More Statements Using do	125
Understanding C's continue Statement	126

Ending a Loop Using C's break Statement	127
Branching with the goto Statement	128
Testing Multiple Conditions	129
Understanding break Within switch	130
Using the switch Statement's default Case	131

MACROS AND CONSTANTS

Defining Constants in Your Programs	132
Understanding Macro and Constant Expansion	133
Naming Constants and Macros	134
Using the <code>__FILE__</code> Preprocessor Constant	135
Using the <code>__LINE__</code> Preprocessor Constant	136
Changing the Preprocessor's Line Count	137
Generating an Unconditional Preprocessor Error	138
Other Preprocessor Constants	139
Recording the Preprocessor Date and Time	140
Testing for ANSI C Compliance	141
Testing for C++ Versus C	142
Undefining a Macro or Constant	143
Comparing Macros and Functions	144
Understanding Compiler Pragmas	145
Learning about Predefined Values and Macros	146
Creating Your Own Header Files	147
Using <code>#include <filename.h></code> OR <code>#include "filename.h"</code>	148
Testing Whether a Symbol is Defined	149
Performing if-else Preprocessing	150
Performing More Powerful Preprocessor Condition Testing	151
Performing if-else and else-if Preprocessing	152
Defining Macros and Constants that Require Multiple Lines ..	153
Creating your own Macros	154
Do Not Place Semicolons in Macro Definitions	155
Creating MIN and MAX Macros	156
Creating SQUARE and CUBE Macros	157
Be Aware of Spaces in Macro Definitions	158
Understanding how to use Parentheses	159
Macros are Typeless	160

UNDERSTANDING STRINGS

Visualizing a C String	161
How the Compiler Represents a Character String	162
How C Stores Character Strings	163
Learning how 'A' Differs from "A"	164
Representing a Quote Within a String Constant	165
Determining the Length of a String	166
Using the <code>strlen</code> Function	167
Copying One String's Characters to Another String	168
Appending One String's Contents to Another String	169
Appending n Characters to a String	170
Transforming One String to Another String	171
Do Not Overwrite a String's Bounds	172
Determining Whether Two Strings are the Same	173
Ignoring Case when Determining Whether Strings are Equal	174
Converting a Character String to Upper- or Lowercase	175
Obtaining the First Occurrence of a Character in a String	176
Returning an Index to the First Occurrence of a String	177
Finding the Last Occurrence of a Character in a String	178
Returning an Index to the Last Occurrence of a String	179
Working with far Strings	180
Writing String Functions for far Strings	181
Counting the Number of Character Occurrences in a String ...	182
Reversing a String's Contents	183
Assigning a Specific Character to an Entire String	184
Comparing Two Character Strings	185
Comparing the First n Characters of Two Strings	186
Comparing Strings Without Considering Case	187
Converting a Character-String Representation of a Number	188
Duplicating a String's Contents	189

Finding a Character from a Given Set's First Occurrence	190
Locating a Substring Within a String	191
Counting the Number of Substring Occurrences	192
Obtaining an Index to a Substring	193
Obtaining the Rightmost Occurrence of a Substring	194
Displaying a String Without the %s Format Specifier	195
Removing a Substring from Within a String	196
Replacing One Substring with Another	197
Converting an ASCII Numeric Representation	198
Determining Whether a Character is Alphanumeric	199
Determining Whether a Character is a Letter	200
Determining Whether a Character Contains an ASCII Value ..	201
Determining Whether a Character is a Control Character	202
Determining Whether a Character is a Digit	203
Determining Whether a Character is a Graphics Character	204
Determining Whether a Character is Upper- or Lowercase	205
Determining Whether a Character is Printable	206
Determining Whether a Character is a Punctuation Symbol ...	207
Determining Whether a Character Contains Whitespace	208
Determining Whether a Character is a Hexadecimal Value	209
Converting a Character to Uppercase	210
Converting a Character to Lowercase	211
Working with ASCII Characters	212
Writing Formatted Output to a String Variable	213
Reading Input from a Character String	214
Tokenizing Strings to Save Space	215
Initializing a String	216

FUNCTIONS

Understanding Functions	217
Using Variables Within Functions	218
Understanding main as a Function	219
Getting Started with Parameters	220
Using Multiple Parameters	221
Understanding Parameter Declarations in Older C Programs ..	222
Returning a Value from a Function	223
Understanding the return Statement	224
Understanding Function Prototypes	225
Understanding the Run-Time Library	226
Understanding Formal and Actual Parameters	227
Resolving Name Conflicts	228
Functions That Do Not Return int	229
Understanding Local Variables	230
How Functions Use the Stack	231
Understanding Function Overhead	232
Understanding Where C Places Local Variables	233
Declaring Global Variables	234
Avoid Using Global Variables	235
Resolving Global and Local Variable Name Conflicts	236
Better Defining a Global Variable's Scope	237
Understanding Call by Value	238
Preventing Parameter Value Change with Call by Value	239
Understanding Call by Reference	240
Getting an Address	241
Using a Variable's Address	242
Changing a Parameter's Value	243
Changing Only Specific Parameters	244
Call by Reference Still Uses the Stack	245
Introducing Function Variables That Remember	246
Understanding How C Initializes Static Variables	247
Using the Pascal Calling Sequence	248
Understanding the Pascal Keyword's Effect	249
Writing a Mixed Language Example	250
Understanding the <code>cdecl</code> Keyword	251
Understanding Recursion	252
Understanding the Recursive factorial Function	253
Programming Another Recursive Example	254

Displaying Values to Better Understand Recursion	255
Understanding Direct and Indirect Recursion	256
Deciding Whether to Use Recursion	257
Understanding Why Recursive Functions are Slow	258
Understanding How to Remove Recursion	259
Passing Strings to Functions	260
Passing Specific Array Elements	261
Understanding const in Formal Parameters	262
Using const Will Not Prevent Parameter Modification	263
Understanding Unbounded String Declarations	264
Using Pointers versus String Declarations	265
How C Uses the Stack for String Parameters	266
Understanding External Variables	267
Putting extern to Use	268
Understanding External static	269
Understanding the volatile Keyword	270
Understanding the Call Frame and Base Pointer	271
Calling an Assembly Language Function	272
Returning a Value from an Assembly Language Function	273
Introducing Functions That Do Not Return Values	274
Understanding Functions That Do Not Use Parameters	275
Understanding the auto Keyword	276
Understanding Scope	277
Understanding C's Categories of Scope	278
Understanding Name Space and Identifiers	279
Understanding Identifier Visibility	280
Understanding Duration	281
Functions That Support a Varying Number of Parameters	282
Supporting a Varying Number of Parameters	283
How va_start, va_arg, and va_end Work	284
Creating Functions That Support Many Parameters and Types	285

KEYBOARD OPERATIONS

Reading a Character from the Keyboard	286
Displaying a Character of Output	287
Understanding Buffered Input	288
Assigning Keyboard Input to a String	289
Combining getch and putchar	290
Remember, getch and putchar are Macros	291
Reading a Character Using Direct I/O	292
Direct Keyboard Input Without Character Display	293
Knowing When to Use '\r' and '\n'	294
Performing Direct Output	295
Placing a Keystroke Back into the Keyboard Buffer	296
Fast Formatted Output Using cprintf	297
Fast Formatted Input from the Keyboard	298
Writing a Character String	299
Faster String Output Using Direct I/O	300
Reading a Character String from the Keyboard	301
Performing Faster Keyboard String Input	302
Displaying Output in Color	303
Clearing the Screen Display	304
Erasing to the End of the Current Line	305
Deleting the Current Screen Line	306
Positioning the Cursor for Screen Output	307
Determining the Row and Column Position	308
Inserting a Blank Line on the Screen	309
Copying Screen Text to a Buffer	310
Writing a Text Buffer at a Specific Screen Location	311
Determining Text Mode Settings	312
Controlling Screen Colors	313
Assigning Background Color	314
Setting the Foreground Color Using textcolor	315
Setting the Background Color Using textbackground	316
Controlling Text Intensity	317
Determining the Current Text Mode	318
Moving Screen Text from One Location to Another	319

Defining a Text Window	320
------------------------------	-----

MATH

Using the Absolute Value of an Integer Expression	321
Using the Arccosine	322
Using the Arcsine	323
Using the Arctangent	324
Obtaining a Complex Number's Absolute Value	325
Rounding Up a Floating-Point Value	326
Using the Cosine of an Angle	327
Using the Hyperbolic Cosine of an Angle	328
Using the Sine of an Angle	329
Using the Hyperbolic Sine of an Angle	330
Using the Tangent of an Angle	331
Using the Hyperbolic Tangent of an Angle	332
Performing Integer Division	333
Working with an Exponential	334
Using the Absolute Value of a Floating-Point Expression	335
Using the Floating-Point Remainder	336
Using a Floating-Point Value's Mantissa and Exponent	337
Calculating the Result of $x * 2e$	338
Calculating the Natural Logarithm	339
Calculating the Result of $\log_{10}x$	340
Determining Maximum and Minimum Values	341
Breaking a double into its Whole and Real Components	342
Calculating the Result of x^n	343
Calculating the Result of $10x$	344
Generating a Random Number	345
Mapping Random Values to a Specific Range	346
Seeding the Random Number Generator	347
Calculating a Value's Square Root	348
Creating a Customized Math Error Handler	349

FILES, DIRECTORIES, AND DISKS

Determining the Current Disk Drive	350
Selecting the Current Drive	351
Determining Available Disk Space	352
Watching Out for dblspace	353
Reading File Allocation Table Information	354
Understanding the Disk ID	355
Performing an Absolute Sector Read or Write	356
Performing BIOS-Based Disk I/O	357
Testing a Floppy Drive's Readiness	358
Opening a File Using fopen	359
Understanding the FILE Structure	360
Closing an Open File	361
Reading and Writing File Information One Character at a Time	362
Understanding the File Pointer's Position Pointer	363
Determining the Current File Position	364
Understanding File Streams	365
Understanding File Translations	366
Understanding the config.sys FILES= Entry	367
Using Low-Level and High-Level File I/O	368
Understanding File Handles	369
Understanding the Process File Table	370
Viewing the Process File Table Entries	371
Understanding the System File Table	372
Displaying the System File Table	373
Deriving File Handles from Stream Pointers	374
Performing Formatted File Output	375
Renaming a File	376
Deleting a File	377
Determining How a Program Can Access a File	378
Setting a File's Access Mode	379
Gaining Better Control of File Attributes	380
Testing for a File Stream Error	381
Determining a File's Size	382

Flushing an I/O Stream	384
Closing All Open Files in One Step	385
Getting a File Stream's File Handle	386
Creating a Temporary Filename Using P_tmpdir	387
Creating a Temporary Filename Using TMP or TEMP	388
Creating a Truly Temporary File	389
Removing Temporary Files	390
Searching the Command Path for a File	391
Searching an Environment Entry's Subdirectory List for a File	391
Opening Files in the TEMP Directory	392
Minimizing File I/O Operations	393
Writing Code that Uses Backslashes in Directory Names	394
Changing the Current Directory	395
Creating a Directory	396
Removing a Directory	397
Removing a Directory Tree	398
Building a Full Pathname	399
Parsing a Directory Path	400
Building a Pathname	401
Opening and Closing a File Using Low-Level Functions	402
Creating a File	403
Performing Low-Level Read and Write Operations	404
Testing for the End of a File	405
Putting the Low-Level File Routines to Work	406
Specifying the Mode for a File-Handle Translation	407
Positioning the File Pointer Using lseek	408
Opening More Than 20 Files	409
Using DOS-Based File Services	410
Obtaining a File's Date and Time Stamp	411
Obtaining a File's Date and Time Using Bit Fields	412
Setting a File's Date and Time Stamp	413
Setting a File Date and Time Stamp to the Current Date and Time	414
Reading and Writing Data One Word at a Time	415
Changing a File's Size	416
Controlling Read and Write File-Open Operations	417
Assigning a File Buffer	418
Allocating a File Buffer	419
Creating a Unique Filename Using mktemp	420
Reading and Writing Structures	421
Reading Structure Data from a File Stream	422
Duplicating a File Handle	423
Forcing a File Handle's Setting	424
Associating a File Handle with a Stream	425
Understanding File Sharing	426
Opening a File for Shared Access	427
Locking a File's Contents	428
Gaining Finer File-Locking Control	429
Working with DOS Directories	430
Opening a Directory	431
Reading a Directory Entry	432
Using Directory Services to Read C:\Windows	433
Rewinding a Directory	434
Reading a Disk's Files Recursively	435
Determining the Current File Position	436
Opening a Shared File Stream	437
Creating a Unique File in a Specific Directory	438
Creating a New File	439
Using the DOS Services to Access a File	440
Forcing a Binary or Text File Open	441
Reading Lines of Text	442
Writing Lines of Text	443
Putting fgets and fputs to Use	444
Forcing Binary File Translation	445
Understanding Why TEXTCOPY Cannot Copy Binary Files	446
Testing for End of File	447
Ungetting a Character	448
Reading Formatted File Data	449

Positioning of the File Pointer Based on its Current Location ..	450
Getting File Handle Information	451
Reopening a File Stream	452

ARRAYS, POINTERS, AND STRUCTURES

Understanding Arrays	453
Declaring an Array	454
Visualizing an Array	455
Understanding an Array's Storage Requirements	456
Initializing an Array	457
Accessing Array Elements	458
Looping Through Array Elements	459
Using Constants to Define Arrays	460
Passing an Array to a Function	461
Revisiting Arrays as Functions	462
Understanding How String Arrays Differ	463
Passing Arrays on the Stack	464
Determining How Many Elements an Array Can Hold	465
Using the Huge Memory Model for Big Arrays	466
The Tradeoff Between Arrays and Dynamic Memory	467
Understanding Multidimensional Arrays	468
Understanding Rows and Columns	469
Accessing Elements in a Two-Dimensional Array	470
Initializing Elements in a Two-Dimensional Array	471
Determining a Multidimensional Array's Memory Consumption	472
Looping Through a Two-Dimensional Array	473
Traversing a Three-Dimensional Array	474
Initializing Multidimensional Arrays	475
Passing a Two-Dimensional Array to a Function	476
Treating Multidimensional Arrays as One Dimensional	477
Understanding How C Stores Multidimensional Arrays	478
Understanding Row-Major Versus Column-Major Order	479
Arrays of Structures of Arrays	480
Understanding Unions	481
Saving Memory with Unions	482
Using REGS—A Classic Union	483
Putting the REGS Union to Use	484
Understanding Bit-Field Structures	485
Visualizing a Bit-Field Structure	486
Understanding a Bitwise Structure's Range of Values	487
Searching an Array for a Specific Value	488
Understanding a Binary Search	489
Using a Binary Search	490
Sorting an Array	491
Understanding the Bubble Sort	492
Putting a Bubble Sort to Use	493
Understanding the Selection Sort	494
Putting a Selection Sort to Use	495
Understanding the Shell Sort	496
Putting a Shell Sort to Use	497
Understanding the Quick Sort	498
Putting a Quick Sort to Use	499
Problems with Previous Sorting Solutions	500
Sorting an Array of Character Strings	501
Searching an Array with lfind	502
Searching for Values with lsearch	503
Searching a Sorted Array with bsearch	504
Sorting Arrays with qsort	505
Determining the Number of Array Elements	506
Understanding Pointers as Addresses	507
Determining a Variable's Address	508
Understanding How C Treats Arrays as Pointers	509
Applying the Address Operator (&) to an Array	510
Declaring Pointer Variables	511
Dereferencing a Pointer	512
Using Pointer Values	513
Using Pointers with Function Parameters	514

Understanding Pointer Arithmetic	515	Understanding Expanded Memory	578
Incrementing and Decrementing a Pointer	516	Using Expanded Memory	579
Combining a Pointer Reference and Increment	517	Understanding Extended Memory	580
Looping Through a String Using a Pointer	518	Understanding Real and Protected Modes	581
Using Functions that Return Pointers	519	Accessing Extended Memory	582
Creating a Function That Returns a Pointer	520	Understanding the High Memory Area	583
Understanding an Array of Pointers	521	Understanding the Stack	584
Visualizing an Array of Character Strings	522	Understanding Different Stack Configurations	585
Looping Through an Array of Character Strings	523	Determining Your Program's Current Stack Size	856
Treating a Character String Array as a Pointer	524	Controlling the Stack Space with <code>_stklen</code>	587
Using a Pointer to a Pointer to Character Strings	525	Assigning a Value to a Memory Range	588
Declaring a String Constant Using a Pointer	526	Copying One Memory Range to Another	589
Understanding the Type <code>void Pointer</code>	527	Copying a Memory Range up to a Specific Byte	590
Creating Pointers to Functions	528	Comparing Two Arrays of unsigned char	591
Using a Pointer to a Function	529	Swapping Adjacent Character String Bytes	592
Using a Pointer to a Pointer to a Pointer	530	Allocating Dynamic Memory	593
Understanding Structures	531	Revisiting Casts	594
A Structure Is a Template for Variable Declarations	532	Releasing Memory When it is no Longer Needed	595
A Structure Tag is the Structure's Name	533	Allocating Memory Using the <code>calloc</code> Function	596
Declaring a Structure Variable in Different Ways	534	Understanding the Heap	597
Understanding Structure Members	535	Getting Around the 64Kb Heap Limit	598
Visualizing a Structure	536	Allocating Memory from the Stack	599
Putting a Structure to Use	537	Allocating Huge Data	600
Passing a Structure to a Function	538	Changing the Size of an Allocated Block	601
Changing a Structure Within a Function	539	Understanding <code>brk</code>	602
Understanding <code>(*pointer).member</code> Indirection	540	Validating the Heap	603
Using the <code>pointer->member</code> Format	541	Performing a Fast Heap Check	604
Using a Tagless Structure	542	Filling Free Heap Space	605
Understanding a Structure Definition's Scope	543	Checking a Specific Heap Entry	606
Initializing a Structure	544	Walking the Heap Entries	607
Performing Structure I/O	545	Peeking into a Specific Memory Location	608
Using a Nested Structure	546	Poking Values into Memory	609
Structures that Contain Arrays	547	Understanding PC Ports	610
Creating an Array of Structures	548	Accessing Port Values	611
DOS AND BIOS SERVICES		Understanding the CMOS	612
Understanding DOS System Services	549	Understanding Memory Models	613
Understanding the BIOS Services	550	Understanding the Tiny Memory Model	614
Understanding Registers	551	Understanding the Small Memory Model	615
Understanding the Flags Register	552	Understanding the Medium Memory Model	616
Understanding Software Interrupts	553	Understanding the Compact Memory Model	617
Using the BIOS to Access the Printer	554	Understanding the Large Memory Model	618
Control+Break Information	555	Understanding the Huge Memory Model	619
Understanding Possible DOS Side Effects	556	Determining the Current Memory Model	620
Suspending a Program Temporarily	557	DATE AND TIME	
Having Some Fun with Sound	558	The Current Date and Time as Seconds Since 1/1/1970	621
Obtaining Country-Specific Information	559	Converting a Date and Time from Seconds to ASCII	622
Understanding the Disk Transfer Address	560	Daylight Savings Adjustment	623
Accessing and Controlling the Disk Transfer Area	561	Delaying a Specific Number of Milliseconds	624
Using the BIOS Keyboard Services	562	Determining Your Program's Processing Time	625
Obtaining the BIOS Equipment List	563	Comparing Two Times	626
Controlling Serial Port I/O	564	Obtaining a Date String	627
Accessing DOS Services Using <code>bdos</code>	565	Obtaining a Time String	628
Getting Extended DOS Error Information	566	Reading the BIOS Timer	629
Determining the BIOS Conventional Memory Amount	567	Working with the Local Time	630
Building a Far Pointer	568	Working with Greenwich Mean Time	631
Breaking a Far Address into a Segment and Offset	569	Getting the DOS System Time	632
Determining Free Core Memory	570	Getting the DOS System Date	633
Reading the Segment Register Settings	571	Setting the DOS System Time	634
MEMORY MANAGEMENT		Setting the DOS System Date	635
Understanding Memory Types	572	Converting a DOS Date to Unix Format	636
Understanding Conventional Memory	573	Using <code>timezone</code> to Compute the Time Zone Difference	637
Understanding the Conventional Memory Layout	574	Determining the Current Time Zone	638
Accessing Conventional Memory	575	Setting Time Zone Fields with <code>tzset</code>	639
Understanding Why the PC and DOS are Restricted to 1Mb	576	Using the TZ Environment Entry	640
Producing an Address from Segments and Offsets	577	Setting the TZ Environment Entry from Within Your Program	641
		Getting Time Zone Information	642

Setting the System Time In Seconds Since Midnight 1/1/1970	643
Converting a Date to Seconds Since Midnight 1/1/1970	644
Determining a Date's Julian Date	645
Creating a Formatted Date and Time String	646
Understanding PC Clock Types	647

REDIRECTING I/O AND PROCESSING COMMAND-LINES

Waiting for a Keypress	648
Prompting the User for a Password	649
Writing Your Own Password Function	650
Understanding Output Redirection	651
Understanding Input Redirection	652
Combining Input and Output Redirection	653
Using stdout and stdin	654
Understanding the Pipe Operator	655
Understanding getchar and putchar	656
Numbering Redirected Input	657
Ensuring That a Message Appears on the Screen	658
Writing Your Own more Command	659
Displaying a Count of Redirected Lines	660
Displaying a Count of Redirected Characters	661
Creating a Timed more Command	662
Preventing I/O Redirection	663
Using the stdprn File Handle	664
Splitting Redirected Output to a File	665
Using the stdaux File Handle	666
Finding Substring Occurrences Within Redirected Input	667
Displaying the First n Lines of Redirected Input	668
Understanding Command-Line Arguments	669
Displaying a Count of Command-Line Arguments	670
Displaying the Command Line	671
Working with Quoted Command-Line Arguments	672
Displaying a File's Contents from the Command Line	673
Treating argv as a Pointer	674
Understanding How C Knows About the Command Line	675
Understanding the Environment	676
Treating env as a Pointer	677
Use void for main's Parameters	678
Working with Command-Line Numbers	679
Understanding Exit Status Values	680
Using return for Exit Status Processing	681
Determining Whether to Declare main as void	682
Searching the Environment for a Specific Entry	683
How DOS Treats the Environment	684
Using the environ Global Variable	685
Adding an Entry to the Current Environment	686
Adding Elements to the DOS Environment	687
Aborting the Current Program	688
Defining Functions That Execute at Program Termination	689

PROGRAMMING TOOLS

Understanding Libraries	690
Reusing Object Code	691
Problems with Compiling C and OBJ Files	692
Creating a Library File	693
Understanding Common Library Operations	694
Listing the Routines in a Library File	695
Use Libraries to Reduce Your Compilation Time	696
Learning More About Your Librarian's Capabilities	697
Understanding the Linker	698
Viewing Linker Capabilities	699
Using a Link Map	700
Using Linker Response Files	701
Simplifying Application Building with MAKE	702
Creating a Simple MAKE File	703
Using Multiple Dependency Files with MAKE	704
Commenting Your MAKE Files	705

Command Lines and MAKE	706
Placing Multiple Dependencies in a MAKE File	707
Explicit and Implicit MAKE Rules	708
Using MAKE Macros	709
Predefined MAKE Macros	710
Performing Conditional Processing with MAKE	711
Testing for a MAKE Macro	712
Including a Second MAKE File	713
Using MAKE's Macro Modifiers	714
Ending a MAKE File with an Error	715
Disabling Command Name Display	716
Using the File BUILTINS.MAK	717
Performing Exit Status Processing in MAKE	718
Invoking and Changing a Macro at the Same Time	719
Executing a MAKE Command for Multiple Dependent Files	720

ADVANCED C

Determining Whether a Math Coprocessor is Present	721
Understanding the ctype.h and istype Macros	722
Controlling Direct Video	723
Detecting System and Math Errors	724
Displaying Predefined Error Messages	725
Determining the Operating System Version Number	726
Understanding Portability	727
Performing a Nonlocal Goto	728
Getting the Process ID (PID)	729
Invoking an Internal DOS Command	730
Using the _psp Global Variable	731
Using the const Modifier in Variable Declarations	732
Using Enumerated Types	733
Putting an Enumerated Type to Use	734
Understanding an Enumerated Value	735
Assigning a Specific Value to an Enumerated Type	736
Saving and Restoring Registers	737
Getting Started with Dynamic Lists	738
Declaring a Linked-List Structure	739
Building a Linked List	740
A Simple Linked-List Example	741
Understanding the Linked-List Traversal	742
Building a More Useful List	743
Appending a List Entry	744
Inserting a List Entry	745
Displaying a Sorted Directory	746
Deleting an Element from a List	747
Using a Doubly Linked List	748
Building a Simple Doubly Linked List	749
Understanding node->previous->next	750
Removing an Element from a Doubly Linked List	751
Inserting an Element into a Doubly Linked List	752
Understanding Child Processes	753
Spawning a Child Process	754
Using Other spawnlxx Functions	755
Using the spawnvxx Functions	756
Execing a Child Process	757
Using Other execlxx Functions	758
Using the execvxx Functions	759
Understanding Overlays	760
Understanding Interrupts	761
The PC Interrupts	762
Using the interrupt Keyword	763
Determining an Interrupt's Vector	764
Setting an Interrupt Vector	765
Enabling and Disabling Interrupts	766
Creating a Simple Interrupt Handler	767
Chaining a Second Interrupt	768
Generating an Interrupt	769
Trapping the PC Timer	770

Understanding Critical Errors	771	Using Bitwise Operators and cout	836
Critical-Error Handling in C.....	772	Understanding Lazy (or Short-Circuit) Evaluation	837
A More Complete Critical-Error Handler	773	Using the const Keyword in C++	838
Restoring Altered Interrupts	774	Using the enum Keyword in C++	839
Creating a Ctrl+Break Handler	775	Understanding Free Space	840
Using DOS Services in Your Critical-Error Handler	776	Allocating Memory with new	841
Improving Performance Using Instruction Set Selection	777	Allocating Multiple Arrays.....	842
Inlining Intrinsic Functions	778	Testing for No Free Space	843
Enabling and Disabling Intrinsic Functions	779	Considerations about Heap Space	844
Understanding Fast Function Calls	780	Using Far Pointers and the new Operator	845
Rules for _fastcall Parameter Passing	781	Releasing Memory Back to the Free Space	846
Understanding Invariant Code	782	Understanding C++ References	847
Understanding Redundant Load Suppression	783	Passing a Reference to a Function	848
Understanding Code Compaction	784	Watching Out for Hidden Objects.....	849
Understanding Loop Compaction	785	Using Three Ways to Pass Parameters	850
Understanding Loop Induction and Strength Reduction	786	Rules for Working with References.....	851
Understanding Common Subexpression Elimination	787	Functions Can Return References	852
Understanding Standard C Conversions	788	Using the C++ inline Keyword	853
Understanding C's Four Basic Types	789	Using the C++ asm Keyword	854
Understanding Fundamental versus Derived Types	790	Reading a Character Using cin	855
Understanding Initializers	791	Writing a Character with cout	856
Understanding Linkage	792	Writing a Simple Filter Program	857
Understanding Tentative Declarations	793	Writing a Simple Tee Command	858
Contrasting Declarations and Definitions	794	Writing a Simple First Command	859
Understanding lvalues	795	Writing a Better First Command.....	860
Understanding rvalues	796	Testing for End of File	861
Using Segment Register Keywords	797	Generating a Newline with endl	862
Use Far Pointers Carefully	798	Understanding Linkage Specifications	863
Understanding Normalized Pointers	799	Understanding Overloading	864
Math Coprocessor Statements	800	Overloading Functions	865
Understanding cdecl and pascal in Variables	801	Overloading Functions: A Second Example	866
Preventing Circular Includes	802	Avoiding Overload Ambiguity	867
GETTING STARTED WITH C++		Reading a Line at a Time with cin	868
Introducing C++	803	Using cin.getline in a Loop	869
How C++ Source Files Differ	804	Changing the new Operator's Default Handling	870
Getting Started with a Simple C++ Program	805	Setting a New Handler with set_new_handler	871
Understanding the cout I/O Stream	806	Determining a C++ Compilation	872
Writing Values and Variables with cout	807	Understanding Structures in C++	873
Combining Different Value Types with cout	808	Introducing Functions as Structure Members.....	874
Displaying Hexadecimal and Octal Values	809	Defining a Member Function Within a Structure	875
Redirecting cout	810	Declaring a Member Function Outside a Structure	876
If You Like printf, Use printf.....	811	Passing Parameters to a Member Function	877
Writing Output to cerr	812	Multiple Variables of the Same Structure	878
Getting Input with cin	813	Different Structures with Same Function Member Names	879
cin Does Not Use Pointers	814	Different Functions with Same Member Names	880
Understanding How cin Selects Data Fields	815	OBJECTS	
Understanding How I/O Streams Know Value Types	816	Understanding Objects	881
Performing Output Using clog	817	Understanding Object-Oriented Programming.....	882
cin, cout, cerr and clog Are Class Instances	818	Understanding Why You Should Use Objects	883
Flushing Output with flush	819	Breaking Programs into Objects	884
Understanding What iostream.h Contains	820	Understanding Objects and Classes	885
C++ Requires Function Prototypes	821	Understanding C++ Classes	886
C++ Adds New Keywords	822	Understanding Encapsulation	887
C++ Supports Anonymous Unions	823	Understanding Polymorphism	888
Resolving Global Scope	824	Understanding Inheritance	889
Providing Default Parameter Values	825	Deciding Between Classes and Structures	890
Controlling cout's Output Width	826	Creating a Simple Class Model	891
Using setw to Set cout Width	827	Implementing the Simple Class Program	892
Specifying a cout Fill Character	828	Defining the Components of a Class.....	893
Right- and Left-Justifying cout Output.....	829	Understanding the Scope Resolution Operator	894
Controlling the Number of Floating-Point Digits cout Displays	830	Using or Omitting the Class Name in Declarations	895
Displaying Values in Fixed or Scientific Format	831	Understanding the public: Label	896
Restoring cout to Default	832	Understanding Information Hiding	897
Setting the I/O Base	833	Understanding the private: Label	898
Declaring Variables Where You Need Them	834	Using the protected: Label.....	899
Placing Default Parameter Values in Function Prototypes	835	Using Public and Private Data	900

Determining What to Hide and What to Make Public	901	Discarding Leading Whitespace on Input	966
Public Methods Are Often Called Interface Functions	902	Understanding Class Libraries	967
Defining Class Functions outside of the Class	903	Place Your Class Definitions in a Header File	968
Defining Methods inside and outside of Classes	904	Using the inline Keyword with Class Member Functions	969
Understanding Object Instances	905	Initializing a Class Array	970
Object Instances Should Share Code	906	Destroying a Class Array	971
Accessing Class Members	907	Creating Initialized Class Arrays	972
Reviewing the Global Resolution Operator	908	Initializing an Array with a Multi-Argument Constructor	973
Initializing Class Values	909	Creating Initialized versus Uninitialized Arrays	974
Using Another Method to Initialize Class Values	910	Working with Class Arrays	975
Understanding Static Class Members	911	Understanding How Class Arrays Use Memory	976
Using Static Data Members	912	Inline Class Code Allows Changes	977
Using Static Member Functions	913	Understanding the Static Store	978
Understanding Member Function Declarations	914		
Using Inline Function Declarations	915		
Determining When to Use Inline and Out-of-Line Functions	916		
Understanding Classes and Unions	917		
Introducing Anonymous Unions	918		
Introducing Friend Functions	919		
Introducing Friend Classes	920		

COMMON CLASS FUNCTIONS

Understanding Constructor Functions	921	Synchronizing I/O Stream Operations with stdio	979
Using Constructor Functions with Parameters	922	Understanding C++ I/O Streams	980
Using a Constructor Function	923	Understanding the C++ Output Streams	981
Understanding When a Program Executes a Constructor	924	Understanding the C++ Input Streams	982
Using Constructor Functions with Parameters	925	Using the ios Members to Format Input and Output	983
Resolving Name Conflicts in a Constructor Function	926	Setting the Format Flags	984
Using a Constructor to Allocate Memory	927	Clearing the Format Flags	985
Handling Memory Allocation in a Cleaner Way	928	Using the Overloaded setf Function	986
Default Parameter Values for Constructors	929	Examining the Current Formatting Flags	987
Overloading Constructor Functions	930	Setting All Flags	988
Finding the Address of an Overloaded Function	931	Using the precision Function	989
Using Constructor Functions with a Single Parameter	932	Using the fill Function	990
Understanding Destructor Functions	933	Understanding Manipulators	991
Using a Destructor Function	934	Using Manipulators to Format I/O	992
Understanding Why You Should Use Destructor Functions	935	Comparing Manipulators and Member Functions	993
Understanding When a Program Invokes a Destructor Function	936	Creating Your Own Inserter Functions	994
Using a Copy Constructor	937	Overloading the Extraction Operator	995
Using Explicit Constructors	938	Another Way to Overload cout's Insertion Operator	996
Understanding Class Scope	939	Creating Your Own Extractor Functions	997
Understanding Nested Classes	940	Creating an Extractor Example	998
Understanding Local Classes	941	Creating Your Own Manipulator Function	999
Resolving Member and Parameter Name Conflicts	942	Creating Parameterless Manipulators	1000
Creating an Array of Class Variables	943	Using Parameters with Manipulators	1001
Constructors and Class Arrays	944	Understanding the Old Stream Class Library	1002
Overloading an Operator	945	Opening a File Stream	1003
Creating a Member operator Function	946	Closing a File Stream	1004
Overloading the Plus Operator	947	Reading and Writing File Stream Data	1005
Overloading the Minus-Sign Operator	948	Checking the Status of a File Operation	1006
Overloading the Prefix and Postfix Increment Operators	949	Putting File Stream Operations Together	1007
Overloading the Prefix and Postfix Decrement Operators	950	Performing a Binary Copy Operation	1008
Revisiting the Restrictions on Operator Overloading	951	Understanding the streambuf Class	1009
Using a friend Function to Overload Operators	952	Writing a Simple streambuf Example	1010
Restrictions on friend Function Operator Overloading	953	Reading Binary Data Using read	1011
Using a friend Function to Overload the ++ or -- Operators	954	Writing Binary Data Using write	1012
Reasons to Overload Operators with friend Functions	955	Using the gcount Member Function	1013
Overloading the new Operator	956	Using the Overloaded get Functions	1014
Overloading the delete Operator	957	Using the getline Method	1015
Overloading new or delete for Arrays	958	Detecting the End of File	1016
Overloading the [] array Operator	959	Using the ignore Function	1017
Overloading the () function call Operator	960	Using the peek Function	1018
Overloading the -> pointer Operator	961	Using the putback Function	1019
Overloading the , comma Operator	962	Finding the Current Position in the File Stream	1020
Understanding Abstraction	963	Controlling the File Stream Pointer	1021
Allocating a Pointer to a Class	964	Using seekg and seekp for Random Access	1022
Discarding a Pointer to a Class	965	Manipulating the File Pointer's Position within a File	1023
		Determining the Current Status of an I/O Stream	1024
		Understanding the Array I/O Classes	1025
		Understanding Character String Streams	1026
		Using istream to Write a Character String	1027
		Better Understanding ostream	1028
		Using the Overloaded istream Forms	1029
		Using pcount with Output Arrays	1030

I/O WITH C++

Manipulating Stream Arrays with the ios Member Functions	1031
Using stringstream	1032
Performing Random Access within a Stream Array	1033
Using Manipulators with Stream Arrays	1034
Using a Custom Insertion Operator with String Arrays	1035
Using Custom Extraction Operators with Stream Arrays	1036
Using Dynamic Arrays with I/O Streams	1037
Understanding the Uses for Stream Array Formatting	1038
Understanding the ends Manipulator	1039
Invoking One Object from Another	1040
Telling the Compiler about a Class	1041
Revisiting friends	1042
Declaring the Reader Class as a friend	1043
Another friend Class Example	1044
Eliminating the Need for the class class_name Statement	1045
Restricting a friend's Access	1046
Name Conflicts and friends	1047

INHERITANCE AND POLYMORPHISM

Inheritance in C++	1048
Understanding Base and Derived Classes	1049
Deriving a Class	1050
Understanding Base and Derived Constructors	1051
Putting Protected Members to Use	1052
Understanding When to Use Protected Members	1053
Reviewing Public and Private Base Class Inheritance	1054
Understanding Protected Base Class Inheritance	1055
Understanding Multiple Inheritance	1056
A Simple Multiple Inheritance	1057
Understanding Constructor Order and Base Classes	1058
Declaring a Base Class as Private	1059
Destructor Functions and Multiple Inheritance	1060
Name Conflicts between Base and Derived Classes	1061
Resolving Class and Base Name Conflicts	1062
Understanding When Inherited Classes Execute Constructors	1063
An Inherited Class Constructor Example	1064
How to Pass Parameters to Base Class Constructors	1065
Understanding Access Declarations with Derived Classes	1066
Using Access Declarations with Derived Classes	1067
Avoiding Ambiguity with Virtual Base Classes	1068
Understanding Virtual Base Classes	1069
Mutual Friends	1070
How A Derived Class Can Become a Base Class	1071
Using Protected Members in Derived Classes	1072
Defining static Class Data	1073
Initializing a static Data Member	1074
Direct Access of a static Data Member	1075
Understanding static private Data Members	1076
Understanding static Member Functions	1077
Direct Access of a public static Function	1078
Using Enhanced Types as Class Members	1079
Nesting a Class	1080
Understanding Subclasses and Superclasses	1081
Inline Assembly Language Statements in a Method Function	1082
Class Members Can Be Recursive	1083
Understanding the this Pointer	1084
How the this Pointer Differs from Other Pointers	1085
Understanding Early and Late Binding	1086
Pointers to Classes	1087
Using the Same Pointer to Different Classes	1088
Base and Derived Name Conflicts with Pointers	1089
Understanding Virtual Functions	1090
Inheriting the Virtual Attribute	1091
Virtual Functions Are Hierarchical	1092
Implementing Polymorphism	1093
Understanding Pure-Virtual functions	1094
Understanding Abstract Classes	1095

Using Virtual Functions	1096
More on Early and Late Binding	1097
Deciding between Early and Late Binding	1098
An Early and Late Binding Example	1099
Defining an Output Stream Manipulator	1100
It is Time to Take a Look at iostream.h	1101
Using sizeof with a Class	1102
Private, Public, and Protected Can Apply to Structures Too	1103
Understanding Class Conversions	1104
Converting Data in a Constructor	1105
Assigning One Class to Another	1106
Use friends for Conversion	1107
Determining When Operators Improve or Reduce Readability	1108

GENERIC FUNCTIONS AND TEMPLATES

Understanding Templates	1109
Putting a Simple Template to Use	1110
Better Understanding Generic Functions	1111
Templates That Support Multiple Types	1112
More on Templates with Multiple Generic Types	1113
Explicitly Overloading a Generic Function	1114
Understanding the Restrictions on Generic Functions	1115
Using a Generic Function	1116
Using a Generic Bubble Sort Function	1117
Using Generic Functions to Compact an Array	1118
Where to Place Templates	1119
Templates Also Eliminate Duplicate Classes	1120
Understanding Generic Classes	1121
Using Generic Classes	1122
Creating a Generic Class with Two Generic Data Types	1123
Creating a Parameterized Manipulator	1124
Creating a Generic Array Class	1125

EXCEPTION HANDLING AND TYPE PORTABILITY

Understanding Exception Handling	1126
Understanding the Basic Exception Handling Form	1127
Writing a Simple Exception Handler	1128
Understanding the throw Statement	1129
Exceptions are Type-Specific	1130
Throwing Exceptions from a Function Within a try Block	1131
Localizing a try Block to a Function	1132
Understanding When the Program Executes catch	1133
Using Multiple catch Statements with a Single try Block	1134
Using the (...) ellipsis Operator with Exceptions	1135
Catching All Exceptions within a Single try Block	1136
Catching Explicit and Generic Exceptions in a Single try Block	1137
Restricting Exceptions	1138
Re-throwing an Exception	1139
Applying Exception Handling	1140
Using Default Function Arguments	1141
Avoiding Errors with Default Function Arguments	1142
Default Arguments versus Function Overloading	1143
Creating Conversion Functions	1144
Using Conversion Functions to Improve Type Portability	1145
Conversion Functions versus Overloaded Operators	1146
Understanding the New C++ Casting Operators	1147
Using the const_cast Operator	1148
Using the dynamic_cast Operator	1149
Using the reinterpret_cast Operator	1150
Using the static_cast Operator	1151
Understanding Namespaces	1152
Using Namespaces	1153
Using the using Statement with namespace	1154
Understanding Run-Time Type Identification	1155
Using typeid for Run-Time Type Identification	1156
Understanding the type_info Class	1157
Understanding the mutable Keyword	1158

Using the mutable Keyword Within a Class	1159
Considerations About the mutable Keyword	1160
Introducing the bool Data Type	1161
Using the bool Data Type	1162

CREATING SAMPLE REUSABLE CLASSES

Creating a String Type	1163
Defining the String Type's Characteristics	1164
Creating the Strings Class	1165
Writing the Constructors for the Strings Class	1166
Performing I/O with the Strings Class	1167
Writing the Assignment Functions for the Strings Class	1168
Overloading the + Operator to Concatenate Strings Objects	1169
Removing a String from Within a Strings Object	1170
Overloading the Relational Operators	1171
Determining a Strings Object's Size	1172
Converting a Strings Object to a Character Array	1173
Using a Strings Object as a Character Array	1174
Demonstrating the Strings Object	1175
Creating a Header for the Strings Class	1176
Another Strings Example	1177
Using a C++ Class to Create a Doubly Linked List	1178
Understanding the dbllinkob Class Members	1179
Understanding the getnext and getprevious Functions	1180
Understanding the Operator Overload Functions	1181
Inheriting the list_object Class	1182
Understanding the linked_list Class	1183
Understanding the linked_list store Function	1184
Understanding the linked_list remove Function	1185
Understanding the getstart and getend Functions	1186
Displaying the linked_list in Forward Order	1187
Displaying the linked_list in Reverse Order	1188
Searching the List	1189
Implementing a Simple linked_list Program	1190
Creating a Generic Doubly Linked List Class	1191
Understanding the Generic list_object Class Members	1192
Understanding the Generic linked_list Class	1193
Using the Generic Classes with a char List	1194
Using the Generic Classes with a double List	1195
Using the Generic Classes with a Structure	1196
Overloading the == Comparison Operator	1197
Other Improvements to the Generic List	1198
Using Objects with the store Function	1199
Writing a Function to Determine the List's Length	1200

STANDARD TEMPLATE LIBRARY

Introducing the Standard Template Library	1201
Understanding the Standard Template Library Header Files ..	1202
Understanding Containers	1203
Using a Container Example	1204
Introducing the Standard Template Library Containers	1205
Understanding Forward and Reversible Containers	1206
Understanding the STL Sequence Containers	1207
Understanding the using namespace std Statement	1208
Understanding the STL Associative Containers	1209
Understanding Iterators	1210
Using an Iterator Example	1211
Better Understanding the STL input and output Iterator Types ..	1212
Understanding the STL's Other Iterator Types	1213
Understanding Concepts	1214
Understanding Models	1215
Understanding Algorithms	1216
Using another STL Algorithm Example	1217
Describing the Algorithms the STL Includes	1218
Studying the STL for_each Algorithm	1219
Studying the STL generate_n Algorithm	1220
Understanding the STL random_shuffle Algorithm	1221
Using the partial_sort_copy Algorithm	1222

Understanding the merge Algorithm	1223
Understanding the inner_product Algorithm	1224
Better Understanding vectors	1225
Using Another Simple vector Program	1226
Comparing vectors to C Arrays	1227
Understanding the bit_vector Sequence Container	1228
Using a Simple bvector Example	1229
Understanding the list Type	1230
Understanding the list Container's Generic Components	1231
Constructing a list Object	1232
Inserting Objects into the List	1233
Using the assign Member Function	1234
Using the remove and empty Member Functions	1235
Traversing the list Object	1236
Understanding the slist Type	1237
Understanding Insertions into an slist Sequence Container ...	1238
Understanding the deque Container	1239
Using the deque Container	1240
Using the erase and clear Member Functions	1241
Using the [] array Operator with a deque	1242
Using reverse Iterators with a deque	1243
Managing the deque's Size	1244
Understanding the map Object	1245
A Simple map Example	1246
Using Member Functions to Manage the map	1247
Controlling the map's Size and Contents	1248
Understanding sets	1249
A Simple set Example	1250

GETTING STARTED WITH WINDOWS PROGRAMMING

Introducing Win32 Programming	1251
More Differences between Windows and DOS Programs	1252
Introducing Threads	1253
Understanding Messages	1254
Understanding a Window's Components	1255
Understanding Parent and Child Windows	1256
Creating a Generic Windows Program	1257
Understanding Resource Files	1258
Understanding Windows Handles	1259
Defining the Windows Handle Types	1260
Understanding the Generic Header File	1261
Understanding Callback Functions	1262
Introducing the Windows Application Programming Interface ..	1263
Looking Closer at the generic.cpp Program	1264
Understanding the WinMain Function	1265
Understanding Window Creation	1266
Understanding the CreateWindow Function	1267
Understanding the ShowWindow Function	1268
Understanding the RegisterClass Function	1269
Learning More about Messages	1270
Using TranslateMessage to Process Messages	1271
Using DispatchMessage to Process Messages	1272
Understanding the Components of a Simple Windows Program	1273
Understanding the LPCTSTR Type	1274
Understanding the DWORD Type	1275
Understanding Windows Pre-Defined Classes	1276
Using Pre-Defined Classes to Create a Simple Window	1277
Windows Sends WM_CREATE When It Creates a Window	1278
Understanding Window and Control Styles	1279
Creating Windows with Extended Styles	1280
Destroying Windows	1281
Understanding the RegisterClassEx API Function	1282
Attaching Information to a Window with SetProp	1283
Using EnumProps to List a Window's Properties	1284
Understanding Callback Functions	1285
Understanding the MessageBox Function	1286
Understanding the MessageBeep Function	1287

MESSAGES AND MENUS

Revisiting Messages	1288
Understanding the Flow of Messages	1289
Better Understanding the MSG Structure's Components	1290
Understanding the PeekMessage Function	1291
Understanding the PostMessage Function	1292
Understanding the SendMessage Function	1293
Using the ReplyMessage Function	1294
Hooking Messages	1295
Using the SetWindowsHookEx Function	1296
Understanding the ExitWindowsEx Function	1297
Understanding Menu Types	1298
Understanding a Menu's Structure	1299
Creating a Menu Within a Resource File	1300
Understanding the POPUP and MENUITEM Descriptors ..	1301
Adding a Menu to an Application's Window	1302
Changing Menus Within an Application	1303
Understanding Menu-Generated Messages	1304
Understanding the LoadMenu Function	1305
Using the ModifyMenu Function	1306
Using EnableMenuItem to Control Menus	1307
Using AppendMenu to Expand a Menu	1308
Using DeleteMenu to Delete Menu Selections	1309
Using Accelerator Keys With Menu Items	1310
Creating a Sample Accelerator Table	1311
Better Understanding the Resource File's Structure	1312
Introducing String Tables	1313
Understanding Custom Resources	1314
Loading String Tables into Programs with LoadString	1315
Listing a Resource File's Contents	1316
Using EnumResourceTypes with Resource Files	1317
Loading Resources into Programs with FindResource	1318

DIALOG BOXES

Understanding Dialog Boxes	1319
Defining Dialog Box Types	1320
Using the Keyboard with Dialog Boxes	1321
Understanding the Components of the Dialog Box Template	1322
Creating a Specific Dialog Box Template	1323
Understanding the Dialog Box Definition's Components	1324
Defining the Dialog Box's Controls	1325
Using the DialogBox Macro to Display a Dialog Box	1326
Understanding the Dialog Box's Message Loop	1327
More on Control Manipulations	1328
Understanding the CreateDialog Macro	1329
Understanding the CreateDialogParam Function	1330
Default Message Processing Within a Dialog Box	1331
Using the DlgDirList Function to Create a Dialog List Box ..	1332
Responding to User Selections Within the List Box	1333
Closing the Dialog Box	1334
Understanding User Input	1335
Responding to Mouse Events	1336
Using the WM_MOUSEMOVE Message	1337
Reading the Mouse Buttons	1338
Responding to Keyboard Events	1339
Understanding Virtual Keys	1340
Using Virtual Keys	1341
More on Using the WM_KEYDOWN Message	1342
Setting and Returning the Mouse's Double-Click Time	1343
Swapping the Mouse Buttons	1344
Determining if the User Has Pressed a Key	1345
Introducing Scroll Bars	1346
Understanding the Different Scroll Bar Types	1347
Using the ShowScrollBar Function	1348
Understanding the Scroll Bar's Position and Range	1349
Understanding the Scroll Bar's Messages	1350
Obtaining the Scroll Bar's Current Settings	1351

Scrolling the Window Content	1352
Understanding the WM_SIZE Message	1353
Understanding the WM_PAINT Message	1354
Other Scroll Bar Messages Your Programs Must Capture	1355
Enabling and Disabling the Scroll Bars	1356
Using the ScrollDC Function	1357

WINDOWS MEMORY MANAGEMENT

Understanding the Win32 Memory Model	1358
Understanding Global and Local Memory	1359
Understanding Virtual Memory	1360
Revisiting Heaps	1361
Allocating a Memory Block from the Global Heap	1362
Using GlobalReAlloc to Dynamically Change Heap Sizes	1363
Discarding an Allocated Memory Block	1364
Using the GlobalFree Function	1365
Using GlobalLock and GlobalHandle	1366
Checking the Computer's Memory	1367
Creating a Heap within a Process	1368
Using Heap Functions to Manage Process-Specific Memory ..	1369
Checking the Size of Memory Allocated from a Heap	1370
Allocating a Virtual Memory Block	1371
Understanding Guard Pages	1372
Better Understanding Virtual Memory Blocks	1373
Freeing up Virtual Memory	1374
Managing Virtual Memory Pages	1375

PROCESSES AND THREADS

Better Understanding Processes	1376
Creating a Process	1377
Terminating Processes	1378
Spawning Child Processes	1379
Working More With Child Processes	1380
Running Detached Child Processes	1381
Better Understanding Threads	1382
Evaluating the Need for Threads	1383
Determining When Not to Create a Thread	1384
Creating a Simple Thread Function	1385
Visualizing the Thread's Startup	1386
Size Steps the Operating System Performs at Thread Creation ...	1387
Determining the Thread's Stack Size	1388
Acquiring a Handle to the Current Thread or Process	1389
Handling the Thread's Processing Time	1390
Managing the Processing Time of Multiple Threads	1391
Better Understanding the GetQueueStatus Function	1392
Handling Unhandled Exceptions	1393
Terminating Threads	1394
Determining the ID of a Thread or Process	1395
Understanding How the Operating System Schedules Threads ..	1396
Introducing Priority Levels	1397
Understanding the Windows Priority Classes	1398
Altering a Process's Priority Class	1399
Setting a Thread's Relative Priority	1400
Obtaining a Thread's Current Priority Level	1401
Obtaining a Thread's Context	1402
Pausing and Resuming Threads	1403
Understanding Thread Synchronization	1404
Defining the Five Major Synchronization Objects	1405
Creating a Critical Section	1406
Using a Simple Critical Section	1407
Using WaitForSingleObject to Synchronize Two Threads	1408
Using WaitForMultipleObjects to Synchronize Many Threads ..	1409
Creating a Mutex	1410
Using a Mutex Within a Sample Program	1411
Using Semaphores	1412
Using a Simple Event Processor	1413

THE GRAPHICS DEVICE INTERFACE

Understanding the Graphics Device Interface	1414
Reasons to Use the Graphics Device Interface	1415
Better Understanding Device Contexts (DC)	1416
Using Private Device Contexts	1417
Understanding Origins and Extents	1418
Obtaining a Device Context to a Window	1419
Creating a Device Context for a Printer	1420
Using CreateCompatibleDC to Create a Memory DC	1421
Understanding the Dangers of CreateDC	1422
Using the CreateFont Function	1423
Using the EnumFontFamilies Function	1424
Displaying Multiple Fonts with CreateFontIndirect	1425
Retrieving a Device's Capabilities	1426
Using the GetSystemMetrics Function to Analyze a Window ..	1427
Understanding the Uses for GetSystemMetrics	1428
Getting a Device Context for an Entire Window	1429
Releasing Device Contexts	1430

XIV BITMAPS, METAFILES, AND ICONS

Getting a Window's Handle from the Device Context	1431
Understanding Device-Dependent Bitmaps	1432
Understanding Device-Independent Bitmaps	1433
Creating Bitmaps	1434
Displaying Bitmaps	1435
Creating DIB Bitmaps	1436
Filling a Rectangle with a Pattern	1437
Using SetDIBits	1438
Using SetDIBitsToDevice to Output to a Given Device	1439
Understanding Metafiles	1440
Creating and Displaying Metafiles	1441
Enumerating the Enhanced Metafiles	1442
Using the GetWinMetaFileBits Function	1443
Understanding Icons	1444
Creating Icons	1445
Creating Icons from a Resource	1446
Using the CreateIconIndirect Function	1447
Using the LoadIcon Function	1448
Using LoadImage to Load Multiple Graphical Types	1449

WINDOWS I/O

Understanding Windows File I/O	1450
Introducing Pipes, Resources, Devices, and Files	1451
Using the CreateFile Function to Open Files	1452
Using CreateFile with Different Devices	1453
Using File Handles	1454
Revisiting File Pointers	1455
Using WriteFile to Write to the File	1456
Using ReadFile to Read from the File	1457
Closing the File	1458
Sharing Data with File Mapping	1459
Mapping a File to Virtual Memory	1460
Mapping a View of a File into the Current Process	1461
Opening a Named File-Mapping Object	1462
Understanding File Attributes	1463
Obtaining and Changing a File's Attributes	1464
Obtaining a File's Size	1465
Obtaining a File's Time Stamp	1466
Creating Directories	1467
Getting and Setting the Current Directory	1468
Getting the Windows and System Directories	1469
Removing Directories	1470
Copying Files	1471
Moving and Renaming Files	1472
Deleting Files	1473
Using FindFirstFile to Locate Files	1474
Using FindNextFile	1475

Closing the Search Handle with FindClose	1476
Searching by Attributes with the FindFile Functions	1477
Using SearchPath Instead of Find to Search	1478
Obtaining a Temporary Path	1479
Creating Temporary Files	1480
Introducing the CreateNamedPipe Function	1481
Connecting a Named Pipe	1482
Calling a Named Pipe	1483
Disconnecting a Named Pipe	1484
Better Understanding Asynchronous Processing	1485
Using Asynchronous Input and Output	1486
Understanding the OVERLAPPED Structure	1487
Asynchronous I/O with a Device Kernel Object	1488
Understanding Working-Set Size Quotas	1489
Setting Higher or Lower Quotas	1490
Understanding the GetLastError Function	1491
Formatting Error Messages with FormatMessage	1492
Asynchronous I/O with an Event Kernel Object	1493
Using WaitForMultipleObjects with Asynchronous I/O	1494
Introducing I/O Completion Ports	1495
Using Alertable I/O for Asynchronous Processing	1496
Alertable I/O Only Works on Windows NT	1497
Using ReadFileEx and WriteFileEx	1498
Using a Callback Completion Routine	1499
Using an Alertable I/O Program	1500

AN INTRODUCTION TO PROGRAMMING

C 1

Computer programs, also known as *software*, are made up of a series of instructions that the computer executes. When you create a program, you must specify the instructions that the computer must execute to perform the desired operations. The process of defining the instructions the computer is to execute is known as *programming*. When you create a program, you store the instructions in an ASCII file whose name usually contains the extension C for a C program and CPP for a C++ program. For example, if you create a C program that performs payroll operations, you might name the file containing the program instructions *payroll.c*. When you create programs, you specify the desired instructions using a *programming language*. C and C++ are only two of many programming languages. Many programmers use programming languages such as BASIC, Pascal, and FORTRAN. Different programming languages provide unique features and have their own strengths (and weaknesses). In any case, programming languages exist to let us define the instructions that we want the computer to execute.

The instructions a computer executes are actually sets of 1's and 0's (binary digits) that represent electronic signals that occur inside the computer. To program the earliest computers (in the 1940s and 1950s), programmers had to understand how the computer interpreted different combinations of 1's and 0's because the programmers wrote all their programs using binary digits. As programs became larger, it became very impractical to make programmers work in terms of the computer's 1's and 0's. Instead, researchers created programming languages that let people express the computer instructions in a form more meaningful to humans. After programmers placed their instructions in a file (called a *source file*), a second program (called a *compiler*), converted the programming language instructions into the 1's and 0's (known as *machine code*) the computer understood. The files on your disk with the EXE and COM extensions contain the machine code (1's and 0's) the computer will execute. Figure 1 illustrates the process of compiling a source code file into an executable program.

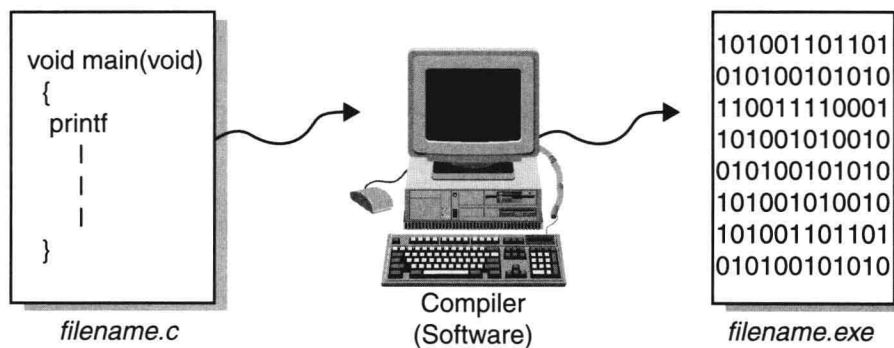


Figure 1 A compiler converts source code instructions into machine code.

After you create a source code file, you run a compiler to convert the instructions into a format the computer can execute. If you are using, for example, Borland's *Turbo C++ Lite™* (included on the companion CD-ROM that accompanies this book), you will use the Compile menu Compile to OBJ option to invoke the compiler (that is, instruct it to compile the source file). The following Tips walk you through the steps required to create and compile a C program.

CREATING AN ASCII SOURCE FILE

C 2

When you create a program, you must place the program statements that you want the computer to execute in a file called a *source file*. If you are not using *Turbo C++ Lite* or a full-featured compiler and editor, you should create your program files using an ASCII editor, such as the EDIT program that DOS provides. You should not create programs using a word processor (such as Microsoft *Word®* or Corel's *WordPerfect®*). As you know, word processors let you format documents by aligning margins, italicizing and underlining text, and so on. To perform these operations, word processors embed special characters within your documents. Although these characters are meaningful to your

word processor, they will confuse the compiler that converts your source file to machine code, and this confusion will cause errors. When you create your source file, make sure you assign a meaningful name that accurately describes the program's function to the file. For example, you might name the source code for a billing program *billing.c*, and the source file for a game program *football.c*.

If, on the other hand, you are using a compiler that includes a built-in editor, you should create your programs within that editor. For example, if you are using *Turbo C++ Lite*, you will use the File menu New option to create a new program file. To create your first program within *Turbo C++ Lite*, perform the following steps:

1. Select the File menu New option. *Turbo C++ Lite* will create the *noname00.cpp* file.
2. Enter the following code into the *noname00.cpp* window:

```
#include <stdio.h>

void main(void)
{
    printf ("Jamsa\'s C/C++ Programmer\'s Bible!");
}
```

3. Select the File menu Save As option. *Turbo C++ Lite* will display the Save File As dialog box.
4. Within the Save File As dialog box, enter the name *first.c* and press ENTER. *Turbo C++ Lite* will save the *first.c* program file.

Although the *first.c* program contains six lines, only the *printf* statement actually performs any work. When you execute this program, *printf* will display the message *Jamsa's C/C++ Programmer's Bible!* on your screen. Every programming language (just like languages such as English, French, and German) has a set of rules, called *syntax rules*, which you must follow when you use the language. When you create C programs, you must obey the syntax rules of the C programming languages. Examples of syntax rules include the parentheses that follow the name *main* and the semicolon at the end of the *printf* instruction. When you type in your program, you must be very careful that you do not omit any of these elements. Double-check your typing to ensure that you have successfully typed in the C program instructions exactly as they appear earlier. If the instructions are correct, save the contents of the file to your disk. In the next Tip you will learn how to compile your source file and to convert your C programming instructions to the machine language that your computer can understand and execute.

3 COMPILING YOUR C PROGRAM

In the previous Tip you created the C source file, *first.c*, which contains the *printf* statement that will display the message *Jamsa's C/C++ Programmer's Bible!* on your screen when you execute the program. A source file contains instructions in a format you can understand (or at least you will be able to understand after you learn C). An executable program, on the other hand, contains instructions expressed as 1's and 0's that the computer understands. The process of converting your C source file to machine code is known as *compiling*. Depending on the C compiler you are using, the command you must perform to compile your source file will differ. Assuming you are using Borland's *Turbo C++ Lite*, you can compile the program (*first.c*) that you created in Tip 2, using the following command sequence:

1. Select the Compile menu Build All option. *Turbo C++ Lite* will display the Compiling dialog box.
2. If the compiler successfully completes the compilation, it will prompt you to *Press any key*. If the C compiler does not create the file *first.exe*, but instead displays error messages on your screen, you have probably violated a C syntax rule, as the next Tip discusses.
3. If you successfully typed in the C statements as shown in Tip 2, the C compiler will create an executable file named *first.exe*. To execute the *first.exe* program, you can either select the Run menu Run option or press the CTRL+F9 keyboard shortcut.