

---

# The Mathematical-Function Computation Handbook

---

Nelson H.F. Beebe

# The Mathematical-Function Computation Handbook

Programming Using the MathCW Portable  
Software Library

Nelson H.F. Beebe  
Department of Mathematics  
University of Utah  
Salt Lake City, UT  
USA

ISBN 978-3-319-64109-6      ISBN 978-3-319-64110-2 (eBook)  
DOI 10.1007/978-3-319-64110-2

Library of Congress Control Number: 2017947446

© Springer International Publishing AG 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature  
The registered company is Springer International Publishing AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Dedication

This book and its software are dedicated to three Williams: Cody, Kahan, and Waite. They taught us that floating-point arithmetic is interesting, intricate, and worth doing right, and showed us how to do it better.

This book and its software are also dedicated to the DEC PDP-10 computer, on which outstanding interactive computing and the Internet were built, and on which this author spent a dozen productive years.

# Preface

WRITE YOUR OWN STORY.  
DON'T LET OTHERS WRITE IT FOR YOU.  
— CHINESE FORTUNE-COOKIE ADVICE.

A *preface* IS GENERALLY SHORTER THAN AN *introduction*, WHICH CONTAINS MATTER KINDRED IN SUBJECT, AND ADDITIONAL OR LEADING UP TO WHAT FOLLOWS; WHILE A *preface* IS USUALLY CONFINED TO PARTICULARS RELATING TO THE ORIGIN, HISTORY, SCOPE, OR AIM OF THE WORK TO WHICH IT IS PREFIXED.

— *New Century Dictionary* (1914).

This book documents a large library that supplies the mathematical functions required by several programming languages, including at least these:

- the 1983 ANSI and 1995 and 2012 ISO Standards [Ada83, Ada95, Ada12] for the Ada programming language;
- the 1990, 1999, and 2011 ISO Standards for C [C90, C99, C11b];
- the 1998, 2003, and 2011 ISO Standards for C++ [C++98, BSI03b, C++03a, C++11a];
- the 2002 and 2006 ECMA and 2006 ISO Standards for C#<sup>®</sup> (pronounced *C-sharp*) [ECM06a, HWG04, CLI05, C#06a, CLI06];
- the 1978 ANSI and 1991, 1997, and 2010 ISO Standards for Fortran [ANSI78, FTN91, FTN97, FTN10];
- the widely used, but not yet standardized, Java<sup>®</sup> programming language [AG96, AG98, CLK99, AGH00, GJSB00, GJSB05, GJS<sup>+</sup>13, GJS<sup>+</sup>14]; and
- the 1990 ISO Extended Pascal Standard [PAS90, JW91].

Numerous scripting languages, including awk, ECMAScript<sup>®</sup>, hoc, JavaScript<sup>®</sup>, Julia, Lua<sup>®</sup>, Perl<sup>®</sup>, PHP, Python<sup>®</sup>, Rexx, Ruby, and Tcl, offer a subset of mathematical functions that are usually drawn from the venerable Fortran and C repertoires. Many other current and historical programming languages, among them Algol 60, Algol 68, COBOL, D, Go, Lisp, Modula, Oberon, OCaml, PL/1, Rust, and Scheme, as well as the Adobe<sup>®</sup> PostScript<sup>®</sup> page-description language, also provide subsets of the Fortran functions.

Although the needs of those languages are a valuable guide for the design of this library, this author's view has been a wider one, and several additional elementary and special functions are provided, including ones for the detection of integer overflow, a sadly neglected area in most architectures and almost all programming languages.

Most current computers, as well as the virtual machines for C#, Java, and PostScript, are based on the *IEEE 754 Standard for Binary Floating-Point Arithmetic* published in 1985, but developed several years earlier, and implemented in hardware in the Intel 8087 coprocessor in 1980. Software intended to be run only on current systems could therefore limit itself to the IEEE 754 architecture. However, three considerations led to the broader view adopted in this book:

- Decimal floating-point arithmetic is part of the 2008 revision of the IEEE 754 Standard [IEEE08, ISO11], and there are proposals to include it in future versions of the ISO C and C++ Standards. Two main reasons to provide decimal arithmetic are its familiarity to humans, and its widespread use in databases.

IBM is a strong proponent of decimal arithmetic, and has produced a firmware implementation on its mainframe systems, and added hardware support in the PowerPC version 6 and later chips. This arithmetic is

based on more than two decades of experience with software decimal arithmetic, notably in the Rexx programming language [Cow85, Cow90] defined by ANSI Standard X3.274-1996 [REXX96], and the NetRexx language [Cow97].

Symbolic-algebra languages, such as Maple<sup>®</sup>, Mathematica<sup>®</sup>, Maxima, MuPAD<sup>®</sup>, PARI/GP, and REDUCE, provide arbitrary-precision arithmetic. The Maple language uses decimal arithmetic, but the others use binary arithmetic.

- There are excellent, and readily available, virtual machines for several historical architectures, including most early microprocessors, the DEC PDP-10, PDP-11, and VAX architectures, the IBM System/360, and even the modern Intel IA-64.

A recent book on virtual machines [SN05] notes that they provide a good way to prototype new machine designs, and have important advantages for security when users can be isolated in private virtual machines.

The great success of the Java programming language has in large part been due to its definition in terms of a virtual machine, called the *Java Virtual Machine (JVM)*<sup>®</sup> [LY97, LY99, SSB01, LYBB13, Set13, LYBB14], and a standard library that provides a uniform environment for Java programs that is independent of the underlying hardware and operating system.

Microsoft's related C# language is also defined in terms of a virtual machine that forms the core of the Microsoft .NET Framework, and the associated free-software reimplementations of .NET in the DotGNU Project<sup>1</sup> and the Mono Project.<sup>2</sup>

Once compilers make programming languages available for the virtual machine, applications, and indeed, even entire operating systems, can be lifted from hardware to virtual machines. This broadens markets for software vendors, removes their dependence on hardware suppliers, and allows them to reach future hardware platforms more quickly.

The commercial Parallels<sup>®</sup>, VirtualBox<sup>®</sup>, Virtual Iron<sup>®</sup>, VMware<sup>®</sup>, and Xen<sup>®</sup> systems have demonstrated that virtual machines adapted to particular hardware can support multiple operating systems on a single CPU with little overhead. Hewlett-Packard, IBM, and Sun Microsystems have all introduced products that allow processors to be logically split and shared among multiple operating systems, or multiple instances of the same operating system. GNU/LINUX *containers* and FreeBSD *jails* provide similar capabilities.

- The Cody and Waite *Software Manual for the Elementary Functions* [CW80] that inspired some of the work described in this book was careful to address the algorithmic issues raised by floating-point designs with number bases other than two. Their book also addresses computations of the elementary functions in *fixed-point* arithmetic, a topic that we largely ignore in this book.

Most books and courses on programming languages spend little time on floating-point arithmetic, so student programmers may consequently conclude that floating-point programming must be trivial, uninteresting, and/or unimportant. This author's view is that floating-point programming does not receive the attention that it deserves, and that there are interesting problems in the design of mathematical software that are not encountered in other types of programming.

However, because a substantial portion of the human population seems to have a fear of mathematics, or at least finds it uncomfortable and unfamiliar, this book takes care to minimize the reader's exposure to mathematics. There are many research papers on the elementary and special functions that provide the mathematical detail that is a necessary foundation for computer implementations, but in most cases, we do not need to deal with it directly. As long as you have some familiarity with programming in at least one common language, and you can recall some of your high-school algebra, and are willing to accept the results of a few short excursions into calculus, you should be able to understand this book, and learn much from it.

On the other hand, professional numerical analysts should find this book of interest as well, because it builds on research by the mathematical software community over the last four decades. In some areas, notably those of converting floating-point numbers to whole numbers, and finding remainders, we present cleaner, simpler, and more portable solutions than are found in existing libraries.

<sup>1</sup>See <http://www.gnu.org/projects/dotgnu/>.

<sup>2</sup>See <http://www.mono-project.com/>.

An important recent research topic has been the development for some of the elementary functions of mathematically proven algorithms that guarantee results that are always correctly rounded. In most cases, these advanced algorithms are quite complex, and beyond the scope of this book. However, for some of the simpler elementary functions, we show how to produce always, or almost always, correctly rounded results.

You might wonder why anyone cares about the correctness of the last few digits of a finite approximation to, say, the square root of 27. Certainly, few humans do. However, it is important to remember that computer hardware, and software libraries, are the foundations of all other software. Numerical computing in particular can be remarkably sensitive to the characteristics and quality of the underlying arithmetic system. By improving the reliability and accuracy of software libraries, we strengthen the foundation. By failing to do so, we leave ourselves open to computational disasters.

## Acknowledgements

This author has had the interest and opportunity to have programmed in scores of programming languages on dozens of operating systems, and most of the major CPU architectures, since his first encounter with the IBM 7044 and IBM System/360 model 50 as an undergraduate student.

At the University of Utah Department of Mathematics, we have a large computing facility that serves thousands of students, as well as departmental staff and faculty, and invited guests. We have made it a point to provide, and preserve, access to a wide range of physical and virtual hardware, and operating systems, to encourage and facilitate software portability testing. A powerful tool, `build-all`, developed by this author and presented in another book [RB05a], makes it easy to automate the building and validating of software packages in parallel on multiple systems.

In recent years, guest access to computing facilities in other locations has been invaluable for development and testing on even more platforms. Those organizations include:

- the University of Utah Center for High-Performance Computing (formerly, the Utah Supercomputing Institute);
- the Henry Eyring Center for Theoretical Chemistry in the Department of Chemistry at the University of Utah;
- the University of Utah Department of Electrical Engineering;
- the University of Utah Department of Physics and Astronomy;
- the Emulab Network Emulation Testbed of the Flux Research Group in the School of Computing at the University of Utah;
- the Hewlett-Packard Test Drive Laboratory;
- the IBM Linux Community Development System; and
- the University of Minnesota Supercomputing Institute.

This author is grateful to the management and staff of all of those excellent facilities, as well as to Intel Corporation for the grant to his department of an IA-64 server.

The mailing list of the IEEE 754 committee, and numerous exchanges with list members, continue to stimulate my interest in floating-point arithmetic. In particular, IBM Fellow Mike Cowlshaw provided useful comments on some of the material on decimal arithmetic in this book, and helped to clarify the history and rationale of decimal arithmetic in computers, and its implementation in modern compilers.

Virtual machines used during the development of this book and its software include:

- the fine commercial VMware system on AMD64, EM64T, and IA-32;
- Ken Harrenstien's outstanding KLH10 implementation of the influential and venerable PDP-10 architecture on which this author worked happily and productively for a dozen years, and the far-ahead-of-its-time DEC TOPS-20 operating system kindly made available to hobbyists by its original vendor;
- Bob Supnik's amazing SIMH simulator of about thirty historical architectures, including the Interdata 8/32 (target of the first port of UNIX from the DEC PDP-11), and the DEC VAX;

- Roger Bowler's Hercules emulator for the IBM System/370, ESA/390, and z/Architecture; and
- The QEMU (Quick EMUlator) hypervisor and KVM (Kernel-based Virtual Machine) that run on most desktop operating systems, including at least Apple MAC OS X and MACOS, DRAGONFLYBSD, FREEBSD, GHOSTBSD, GNU/LINUX, HAIKU, HARDENEDBSD, MIDNIGHTBSD, Microsoft WINDOWS, NETBSD, OPENBSD, PAC BSD, PCBSD, OPENSOLARIS, REACTOS, and TRUEOS.

This book, and all of the software that it describes, is a single-person production by this author. Years of following the scientific research literature, and published books, in broad areas of computer science, theoretical chemistry, computational physics, and numerical mathematics, and a lifetime spent in programming in scientific computing and other areas, gave the experience needed to tackle a project with the breadth of scope of the software library described here.

## The Unix family

The Open Group currently owns the all-caps registered trademark UNIX<sup>®</sup> for a descendant of the operating system developed at AT&T Bell Laboratories starting about 1969, and permits vendors whose implementations pass an extensive test suite to license use of that trademark. Because of legal wrangling, ownership of the specific name UNIX was long disputed in US courts. That led computer manufacturers to rebrand their customized versions under other trademarked names, and similarly, in the free-software world, each O/S distribution seems to acquire its own unique name. In this book, we use the capitalized name UNIX to refer to the entire family.

The GNU<sup>®</sup> system, where *GNU* stands for the infinitely recursive phrase *GNU is Not Unix*, is, for the purposes of this book, also a UNIX-like system. If this matters to you, just mentally add the suffix *-like* to every mention of UNIX in this book.

Filesystem implementations, operating-system kernel details, software packaging practices, and system-management procedures and tools, differ, sometimes dramatically so, across different members of the UNIX family. However, for the ordinary *user* of those systems, they are all familiar and similar, because most of their commonly used commands are nearly identical in behavior and name. The notions of files as byte streams, devices, kernel data, and networks treated as files, command shells for interactive use and scripting, simple I/O redirection mechanisms, and pipes for easy connection of many smaller programs into larger, and more powerful, ones, are critical features of the UNIX environment. A common window system, X11, provides mechanism but not policy, and separates program, window display, and window management, allowing each to run on separate computers if desired, but joined by secure encrypted communications channels.

Many UNIX distributions have long been available as open-source software, allowing programmers all over the world to contribute documentation, enhancements, and sometimes, radical new ideas. Internet mailing lists allow them to keep in regular contact and develop long-time electronic friendships, even though they may never have a chance to meet in person. The UNIX world is truly a global community.

## Trademarks, copyrights, and property ownership

Because it deals with real computers, operating systems, and programming languages, both historic and current, this book necessarily contains many references to names that are copyrighted, registered, or trademarked, and owned by various firms, foundations, organizations, and people. Except in this preface, we do not clutter the book text with the traditional superscript symbols that mark such ownership, but we acknowledge it here, and we index every reference to model, product, and vendor names. Among the commercial software systems most frequently mentioned in this book are Maple, Mathematica, MATLAB<sup>®</sup>, and MuPAD. In general, we should expect almost any commercial entity, or commercial product, to have a name that is registered or trademarked. The computing industry that has so changed human history is the work of many companies and many people from many countries, and we should view it proudly as part of our shared modern heritage.

The International Organization for Standardization (ISO) kindly gave permission to cite in this book (usually brief) snippets of portions of the ISO Standards for the C language. Because of the language precision in Standards, it is essential for correct software development to be guided by the original exact wording, rather than working from existing practice, hearsay, imperfect human memory, or paraphrased text.



## To show code, or not

Authors of books about software have to make a choice of whether to show actual code in a practical programming language, or only to provide imprecise descriptions in flowcharts or pseudocode.

Although the latter approach seems to be more common, it has the serious drawback that the code cannot be tested by a compiler or a computer, and long experience says that untested code is certain to contain bugs, omissions, and pitfalls. Writing software is hard, writing floating-point software is harder yet, and writing such software to work correctly on a broad range of systems is even more difficult. Software often outlives hardware, so portability is important.

There are many instances in this book where subtle issues arise that *must* be properly handled in software, and the only way to do so is to use a real programming language where the code can undergo extensive testing. The floating-point programmer must have broad experience with many architectures, because obscure architectural assumptions and platform dependencies can all too easily riddle code, making it much less useful than it could be, if more care were paid to its design.

This author has therefore chosen to show actual code in many sections of this book, and to index it thoroughly, but not to show all of the code, or even all of the commentary that is part of it. Indeed, a simple printed listing of the code in a readable type size is several times longer than this book.

When the code is routine, as it is for much of the validation testing and interfacing to other programming languages, there is little point in exhibiting it in detail. All of the code is freely available for inspection, use, and modification by the reader anyway, because it is easily accessible online. However, for many shorter routines, and also for complicated algorithms, it is instructive to display the source code, and describe it in prose.

As in most human activities, programmers learn best by hands-on coding, but they can often learn as much, or more, by reading well-written code produced by others. It is this author's view that students of programming can learn a lot about the subject by working through a book like this, with its coverage of a good portion of the run-time library requirements of one of the most widely used programming languages. This is *real* code intended for *real* work, for portable and reliable operation, and for influencing the future development of programming languages to support more numeric data types, higher precision, more functionality, and increased dependability and security.

## To cite references, or not

Although research articles are generally expected to contain copious citations of earlier work, textbooks below the level of graduate research may be devoid of references. In this book, we take an intermediate approach: citations, when given, are a guide to further study that the reader may find helpful and interesting. When the citations are to research articles or reports, the bibliography entries contain Web addresses for documents that could be located in online archives at the time of writing this book.

Until the advent of computers, tracking research publications was difficult, tedious, and time consuming. Access to new journal issues often required physical visits to libraries. Photocopiers made it easier to abandon the long-time practice of sending postcards to authors with requests for article reprints. The world-wide Internet has changed that, and many journals, and some books, now appear only electronically. Internet search engines often make it possible to find material of interest, but the search results still require a lot of labor to record and re-use, and also to validate, because in practice, search results are frequently incomplete and unreliable. Most journal publishers have Web sites that provide contents information for journal volumes, but there is no common presentation format. Also, a search of one publisher's site is unlikely to find related works in journals produced by its competitors.

The BibTeX bibliography markup system developed by Oren Patashnik at Stanford University as part of the decade-long TeX Project has provided an important solution to the problem of making publication information *reusable*. This author has expended considerable effort in writing software to convert library and publisher data into BibTeX form, and developed many tools for checking, curating, ordering, searching, sorting, and validating BibTeX data.

Two freely available collections, the *BibNet Project* archives and the *TeX User Group* archives, both hosted at the author's university, and mirrored to other sites, provide a view into the research literature of selected areas of chemistry, computer science, computer standards, cryptography, mathematics, physics, probability and statistics, publication metrics, typesetting, and their associated histories.

Those archives provide a substantial pool of data from which specialized collections, such as the bibliography for this book, can be relatively easily derived, *without* retyping publication data. BibTeX supports citation, cross

referencing, extraction, sorting, and formatting of publication data in hundreds of styles. Additional software written by this author extends B $\text{\LaTeX}$  by automatically producing the separate author/editor index that appears in the back matter of this book, and enhancing bibliography entries with lists of page numbers where each is cited. Thus, a reader who remembers *just one author* of a cited work can quickly find both the reference, and the locations in the book where it is cited.

## The MathCW Web site

This book is accompanied by a Web site maintained by this author at

<http://www.math.utah.edu/pub/mathcw/>

That site contains

- source code for the book's software;
- a B $\text{\LaTeX}$  database, `mathcw.bib`, from a subset of which all references that appear in the bibliography in this book's back matter are automatically derived and formatted;
- related materials developed after the book has been frozen for publication;
- compiled libraries for numerous systems; and
- pre-built C compilers with support for decimal arithmetic on several operating systems.

It is expected to be mirrored to many other sites around the world, to ensure wide availability, and protection against loss.

The mathcw software is released in versioned bundles, and its history is maintained in a revision control system to preserve a record of its development and future evolution, as is common with most large modern software projects.

# Contents

<b>List of figures</b>	<b>xxv</b>
<b>List of tables</b>	<b>xxxi</b>
<b>Quick start</b>	<b>xxxv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Programming conventions . . . . .	2
1.2 Naming conventions . . . . .	4
1.3 Library contributions and coverage . . . . .	5
1.4 Summary . . . . .	6
<b>2 Iterative solutions and other tools</b>	<b>7</b>
2.1 Polynomials and Taylor series . . . . .	7
2.2 First-order Taylor series approximation . . . . .	8
2.3 Second-order Taylor series approximation . . . . .	9
2.4 Another second-order Taylor series approximation . . . . .	9
2.5 Convergence of second-order methods . . . . .	10
2.6 Taylor series for elementary functions . . . . .	10
2.7 Continued fractions . . . . .	12
2.8 Summation of continued fractions . . . . .	17
2.9 Asymptotic expansions . . . . .	19
2.10 Series inversion . . . . .	20
2.11 Summary . . . . .	22
<b>3 Polynomial approximations</b>	<b>23</b>
3.1 Computation of odd series . . . . .	23
3.2 Computation of even series . . . . .	25
3.3 Computation of general series . . . . .	25
3.4 Limitations of Cody/Waite polynomials . . . . .	28
3.5 Polynomial fits with Maple . . . . .	32
3.6 Polynomial fits with Mathematica . . . . .	33
3.7 Exact polynomial coefficients . . . . .	42
3.8 Cody/Waite rational polynomials . . . . .	43
3.9 Chebyshev polynomial economization . . . . .	43
3.10 Evaluating Chebyshev polynomials . . . . .	48
3.11 Error compensation in Chebyshev fits . . . . .	50
3.12 Improving Chebyshev fits . . . . .	51
3.13 Chebyshev fits in rational form . . . . .	52
3.14 Chebyshev fits with Mathematica . . . . .	56
3.15 Chebyshev fits for function representation . . . . .	57
3.16 Extending the library . . . . .	57
3.17 Summary and further reading . . . . .	58

<b>4</b>	<b>Implementation issues</b>	<b>61</b>
4.1	Error magnification . . . . .	61
4.2	Machine representation and machine epsilon . . . . .	62
4.3	IEEE 754 arithmetic . . . . .	63
4.4	Evaluation order in C . . . . .	64
4.5	The <code>volatile</code> type qualifier . . . . .	65
4.6	Rounding in floating-point arithmetic . . . . .	66
4.7	Signed zero . . . . .	69
4.7.1	Detecting the sign of zero . . . . .	69
4.7.2	Signed-zero constants . . . . .	69
4.7.3	Arc tangent and signed zero . . . . .	70
4.8	Floating-point zero divide . . . . .	70
4.9	Floating-point overflow . . . . .	71
4.10	Integer overflow . . . . .	72
4.10.1	Preventing integer overflow . . . . .	74
4.10.1.1	Safe integer absolute value . . . . .	74
4.10.1.2	Safe integer addition . . . . .	75
4.10.1.3	Safe integer division . . . . .	75
4.10.1.4	Safe integer multiplication . . . . .	75
4.10.1.5	Safe integer negation . . . . .	76
4.10.1.6	Safe integer remainder . . . . .	76
4.10.1.7	Safe integer subtraction . . . . .	76
4.10.1.8	Safe integer operations: a retrospective . . . . .	77
4.11	Floating-point underflow . . . . .	77
4.12	Subnormal numbers . . . . .	78
4.13	Floating-point inexact operation . . . . .	79
4.14	Floating-point invalid operation . . . . .	79
4.15	Remarks on NaN tests . . . . .	80
4.16	UlpS — units in the last place . . . . .	81
4.17	Fused multiply-add . . . . .	85
4.18	Fused multiply-add and polynomials . . . . .	88
4.19	Significance loss . . . . .	89
4.20	Error handling and reporting . . . . .	89
4.21	Interpreting error codes . . . . .	93
4.22	C99 changes to error reporting . . . . .	94
4.23	Error reporting with threads . . . . .	95
4.24	Comments on error reporting . . . . .	95
4.25	Testing function implementations . . . . .	96
4.25.1	Taylor-series evaluation . . . . .	97
4.25.2	Argument purification . . . . .	97
4.25.3	Addition-rule evaluation . . . . .	98
4.25.4	Relative error computation . . . . .	99
4.26	Extended data types on Hewlett–Packard HP-UX IA-64 . . . . .	100
4.27	Extensions for decimal arithmetic . . . . .	101
4.28	Further reading . . . . .	103
4.29	Summary . . . . .	104
<b>5</b>	<b>The floating-point environment</b>	<b>105</b>
5.1	IEEE 754 and programming languages . . . . .	105
5.2	IEEE 754 and the <code>mathcw</code> library . . . . .	106
5.3	Exceptions and traps . . . . .	106
5.4	Access to exception flags and rounding control . . . . .	107
5.5	The environment access pragma . . . . .	110
5.6	Implementation of exception-flag and rounding-control access . . . . .	110

5.6.1	Clearing exception flags: <code>feclearexcept()</code> . . . . .	110
5.6.2	Getting the rounding direction: <code>fegetround()</code> . . . . .	111
5.6.3	Raising exception flags: <code>feraiseexcept()</code> . . . . .	111
5.6.4	Setting the rounding direction: <code>fesetround()</code> . . . . .	111
5.6.5	Testing exception flags: <code>fetestexcept()</code> . . . . .	112
5.6.6	Comments on the core five . . . . .	112
5.7	Using exception flags: simple cases . . . . .	112
5.8	Using rounding control . . . . .	115
5.9	Additional exception flag access . . . . .	116
5.9.1	Getting the environment: <code>fegetenv()</code> . . . . .	117
5.9.2	Setting the environment: <code>fesetenv()</code> . . . . .	117
5.9.3	Getting exception flags: <code>fegetexceptflag()</code> . . . . .	118
5.9.4	Setting exception flags: <code>fesetexceptflag()</code> . . . . .	118
5.9.5	Holding exception flags: <code>fehldexcept()</code> . . . . .	119
5.9.6	Updating the environment: <code>feupdateenv()</code> . . . . .	119
5.9.7	Comments on the six functions . . . . .	120
5.10	Using exception flags: complex case . . . . .	120
5.11	Access to precision control . . . . .	123
5.11.1	Precision control in hardware . . . . .	124
5.11.2	Precision control and the AMD64 architecture . . . . .	124
5.11.3	Precision control in the <code>mathcw</code> library . . . . .	124
5.12	Using precision control . . . . .	126
5.13	Summary . . . . .	127
<b>6</b>	<b>Converting floating-point values to integers</b> . . . . .	<b>129</b>
6.1	Integer conversion in programming languages . . . . .	129
6.2	Programming issues for conversions to integers . . . . .	130
6.3	Hardware out-of-range conversions . . . . .	131
6.4	Rounding modes and integer conversions . . . . .	132
6.5	Extracting integral and fractional parts . . . . .	132
6.6	Truncation functions . . . . .	135
6.7	Ceiling and floor functions . . . . .	136
6.8	Floating-point rounding functions with fixed rounding . . . . .	137
6.9	Floating-point rounding functions: current rounding . . . . .	138
6.10	Floating-point rounding functions without <i>inexact</i> exception . . . . .	139
6.11	Integer rounding functions with fixed rounding . . . . .	140
6.12	Integer rounding functions with current rounding . . . . .	142
6.13	Remainder . . . . .	143
6.14	Why the remainder functions are hard . . . . .	144
6.15	Computing <code>fmod()</code> . . . . .	146
6.16	Computing <code>remainder()</code> . . . . .	148
6.17	Computing <code>remquo()</code> . . . . .	150
6.18	Computing one remainder from the other . . . . .	152
6.19	Computing the remainder in nonbinary bases . . . . .	155
6.20	Summary . . . . .	156
<b>7</b>	<b>Random numbers</b> . . . . .	<b>157</b>
7.1	Guidelines for random-number software . . . . .	157
7.2	Creating generator seeds . . . . .	158
7.3	Random floating-point values . . . . .	160
7.4	Random integers from floating-point generator . . . . .	165
7.5	Random integers from an integer generator . . . . .	166
7.6	Random integers in ascending order . . . . .	168
7.7	How random numbers are generated . . . . .	169
7.7.1	Linear congruential generators . . . . .	169

7.7.2	Deficiencies of congruential generators . . . . .	170
7.7.3	Computing congruential generators . . . . .	171
7.7.4	Faster congruential generators . . . . .	174
7.7.5	Other generator algorithms . . . . .	176
7.7.6	Combined generators . . . . .	177
7.7.7	Cryptographic generators . . . . .	178
7.8	Removing generator bias . . . . .	178
7.9	Improving a poor random number generator . . . . .	178
7.10	Why long periods matter . . . . .	179
7.11	Inversive congruential generators . . . . .	180
7.11.1	Digression: Euclid's algorithm . . . . .	180
7.11.1.1	Euclid's algorithm for any integers . . . . .	183
7.11.1.2	Division-free gcd algorithm . . . . .	184
7.11.2	Another digression: the extended Euclid's algorithm . . . . .	186
7.12	Inversive congruential generators, revisited . . . . .	189
7.13	Distributions of random numbers . . . . .	189
7.13.1	The uniform distribution . . . . .	189
7.13.2	The exponential distribution . . . . .	189
7.13.3	The logarithmic distribution . . . . .	190
7.13.4	The normal distribution . . . . .	192
7.13.5	More on the normal distribution . . . . .	194
7.14	Other distributions . . . . .	195
7.14.1	Numerical demonstration of the Central Limit Theorem . . . . .	196
7.15	Testing random-number generators . . . . .	196
7.15.1	The chi-square test . . . . .	197
7.15.2	Random-number generator test suites . . . . .	200
7.15.3	Testing generators for nonuniform distributions . . . . .	202
7.16	Applications of random numbers . . . . .	202
7.16.1	Monte Carlo quadrature . . . . .	202
7.16.2	Encryption and decryption . . . . .	203
7.16.2.1	Problems with cryptography . . . . .	203
7.16.2.2	A provably secure encryption method . . . . .	204
7.16.2.3	Demonstration of a one-time pad . . . . .	204
7.16.2.4	Attacks on one-time-pad encryption . . . . .	207
7.16.2.5	Choice of keys and encryption methods . . . . .	207
7.16.2.6	Caveats about cryptography . . . . .	208
7.17	The mathcw random number routines . . . . .	208
7.18	Summary, advice, and further reading . . . . .	214
<b>8</b>	<b>Roots</b> . . . . .	<b>215</b>
8.1	Square root . . . . .	215
8.1.1	Considerations for rounding of the square root . . . . .	216
8.1.2	An algorithm for correct rounding of the square root . . . . .	217
8.1.3	Variant iterations for the square root . . . . .	220
8.2	Hypotenuse and vector norms . . . . .	222
8.3	Hypotenuse by iteration . . . . .	227
8.4	Reciprocal square root . . . . .	233
8.4.1	Improved rounding of the reciprocal square root . . . . .	234
8.4.2	Almost-correct rounding of the reciprocal square root . . . . .	235
8.5	Cube root . . . . .	237
8.5.1	Improved rounding of the cube root . . . . .	237
8.5.2	Almost-correct rounding of the cube root . . . . .	239
8.6	Roots in hardware . . . . .	240
8.7	Summary . . . . .	242

<b>9</b>	<b>Argument reduction</b>	<b>243</b>
9.1	Simple argument reduction . . . . .	243
9.2	Exact argument reduction . . . . .	250
9.3	Implementing exact argument reduction . . . . .	253
9.4	Testing argument reduction . . . . .	265
9.5	Retrospective on argument reduction . . . . .	265
<b>10</b>	<b>Exponential and logarithm</b>	<b>267</b>
10.1	Exponential functions . . . . .	267
10.2	Exponential near zero . . . . .	273
10.3	Logarithm functions . . . . .	282
10.3.1	Computing logarithms in a binary base . . . . .	284
10.3.2	Computing logarithms in a decimal base . . . . .	287
10.4	Logarithm near one . . . . .	290
10.5	Exponential and logarithm in hardware . . . . .	292
10.6	Compound interest and annuities . . . . .	294
10.7	Summary . . . . .	298
<b>11</b>	<b>Trigonometric functions</b>	<b>299</b>
11.1	Sine and cosine properties . . . . .	299
11.2	Tangent properties . . . . .	302
11.3	Argument conventions and units . . . . .	304
11.4	Computing the cosine and sine . . . . .	306
11.5	Computing the tangent . . . . .	310
11.6	Trigonometric functions in degrees . . . . .	313
11.7	Trigonometric functions in units of $\pi$ . . . . .	315
11.7.1	Cosine and sine in units of $\pi$ . . . . .	316
11.7.2	Cotangent and tangent in units of $\pi$ . . . . .	318
11.8	Computing the cosine and sine together . . . . .	320
11.9	Inverse sine and cosine . . . . .	323
11.10	Inverse tangent . . . . .	331
11.11	Inverse tangent, take two . . . . .	336
11.12	Trigonometric functions in hardware . . . . .	338
11.13	Testing trigonometric functions . . . . .	339
11.14	Retrospective on trigonometric functions . . . . .	340
<b>12</b>	<b>Hyperbolic functions</b>	<b>341</b>
12.1	Hyperbolic functions . . . . .	341
12.2	Improving the hyperbolic functions . . . . .	345
12.3	Computing the hyperbolic functions together . . . . .	348
12.4	Inverse hyperbolic functions . . . . .	348
12.5	Hyperbolic functions in hardware . . . . .	350
12.6	Summary . . . . .	352
<b>13</b>	<b>Pair-precision arithmetic</b>	<b>353</b>
13.1	Limitations of pair-precision arithmetic . . . . .	354
13.2	Design of the pair-precision software interface . . . . .	355
13.3	Pair-precision initialization . . . . .	356
13.4	Pair-precision evaluation . . . . .	357
13.5	Pair-precision high part . . . . .	357
13.6	Pair-precision low part . . . . .	357
13.7	Pair-precision copy . . . . .	357
13.8	Pair-precision negation . . . . .	358
13.9	Pair-precision absolute value . . . . .	358
13.10	Pair-precision sum . . . . .	358

13.11	Splitting numbers into pair sums	359
13.12	Premature overflow in splitting	362
13.13	Pair-precision addition	365
13.14	Pair-precision subtraction	367
13.15	Pair-precision comparison	368
13.16	Pair-precision multiplication	368
13.17	Pair-precision division	371
13.18	Pair-precision square root	373
13.19	Pair-precision cube root	377
13.20	Accuracy of pair-precision arithmetic	379
13.21	Pair-precision vector sum	384
13.22	Exact vector sums	385
13.23	Pair-precision dot product	385
13.24	Pair-precision product sum	386
13.25	Pair-precision decimal arithmetic	387
13.26	Fused multiply-add with pair precision	388
13.27	Higher intermediate precision and the FMA	393
13.28	Fused multiply-add without pair precision	395
13.29	Fused multiply-add with multiple precision	401
13.30	Fused multiply-add, Boldo/Melquiond style	403
13.31	Error correction in fused multiply-add	406
13.32	Retrospective on pair-precision arithmetic	407
<b>14</b>	<b>Power function</b>	<b>411</b>
14.1	Why the power function is hard to compute	411
14.2	Special cases for the power function	412
14.3	Integer powers	414
14.4	Integer powers, revisited	420
14.5	Outline of the power-function algorithm	421
14.6	Finding $a$ and $p$	423
14.7	Table searching	424
14.8	Computing $\log_n(g/a)$	426
14.9	Accuracy required for $\log_n(g/a)$	429
14.10	Exact products	430
14.11	Computing $w$ , $w_1$ and $w_2$	433
14.12	Computing $n^{w_2}$	437
14.13	The choice of $q$	438
14.14	Testing the power function	438
14.15	Retrospective on the power function	440
<b>15</b>	<b>Complex arithmetic primitives</b>	<b>441</b>
15.1	Support macros and type definitions	442
15.2	Complex absolute value	443
15.3	Complex addition	445
15.4	Complex argument	445
15.5	Complex conjugate	446
15.6	Complex conjugation symmetry	446
15.7	Complex conversion	448
15.8	Complex copy	448
15.9	Complex division: C99 style	449
15.10	Complex division: Smith style	451
15.11	Complex division: Stewart style	452
15.12	Complex division: Priest style	453
15.13	Complex division: avoiding subtraction loss	455
15.14	Complex imaginary part	456



15.15	Complex multiplication . . . . .	456
15.16	Complex multiplication: error analysis . . . . .	458
15.17	Complex negation . . . . .	459
15.18	Complex projection . . . . .	460
15.19	Complex real part . . . . .	460
15.20	Complex subtraction . . . . .	461
15.21	Complex infinity test . . . . .	462
15.22	Complex NaN test . . . . .	462
15.23	Summary . . . . .	463
<b>16</b>	<b>Quadratic equations</b>	<b>465</b>
16.1	Solving quadratic equations . . . . .	465
16.2	Root sensitivity . . . . .	471
16.3	Testing a quadratic-equation solver . . . . .	472
16.4	Summary . . . . .	474
<b>17</b>	<b>Elementary functions in complex arithmetic</b>	<b>475</b>
17.1	Research on complex elementary functions . . . . .	475
17.2	Principal values . . . . .	476
17.3	Branch cuts . . . . .	476
17.4	Software problems with negative zeros . . . . .	478
17.5	Complex elementary function tree . . . . .	479
17.6	Series for complex functions . . . . .	479
17.7	Complex square root . . . . .	480
17.8	Complex cube root . . . . .	485
17.9	Complex exponential . . . . .	487
17.10	Complex exponential near zero . . . . .	492
17.11	Complex logarithm . . . . .	495
17.12	Complex logarithm near one . . . . .	497
17.13	Complex power . . . . .	500
17.14	Complex trigonometric functions . . . . .	502
17.15	Complex inverse trigonometric functions . . . . .	504
17.16	Complex hyperbolic functions . . . . .	509
17.17	Complex inverse hyperbolic functions . . . . .	514
17.18	Summary . . . . .	520
<b>18</b>	<b>The Greek functions: gamma, psi, and zeta</b>	<b>521</b>
18.1	Gamma and log-gamma functions . . . . .	521
18.1.1	Outline of the algorithm for <code>tgamma()</code> . . . . .	525
18.1.1.1	Asymptotic expansions . . . . .	528
18.1.1.2	Recurrence-relation accuracy . . . . .	528
18.1.1.3	Sums of rational numbers . . . . .	529
18.1.1.4	Avoiding catastrophic overflow . . . . .	530
18.1.2	Gamma function accuracy . . . . .	531
18.1.3	Computation of $\pi / \sin(\pi x)$ . . . . .	531
18.1.4	Why <code>lgamma(x)</code> is hard to compute accurately . . . . .	531
18.1.5	Outline of the algorithm for <code>lgamma()</code> . . . . .	534
18.1.6	Log-gamma function accuracy . . . . .	536
18.2	The <code>psi()</code> and <code>psiln()</code> functions . . . . .	536
18.2.1	Psi function poles and zeros . . . . .	538
18.2.2	Recurrence relations for psi functions . . . . .	539
18.2.3	Psi functions with negative arguments . . . . .	541
18.2.4	Psi functions with argument multiples . . . . .	541
18.2.5	Taylor-series expansions of psi functions . . . . .	542
18.2.6	Asymptotic expansion of the psi function . . . . .	542

18.2.7	Psi function on $[0, 1]$ . . . . .	543
18.2.8	Outline of the algorithm for $\text{psi}()$ . . . . .	543
18.2.9	Computing $\pi / \tan(\pi x)$ . . . . .	545
18.2.10	Outline of the algorithm for $\text{psi} \ln()$ . . . . .	546
18.3	Polygamma functions . . . . .	547
18.3.1	Applications of polygamma functions . . . . .	555
18.3.2	Computing the polygamma functions . . . . .	556
18.3.3	Retrospective on the polygamma functions . . . . .	558
18.4	Incomplete gamma functions . . . . .	560
18.5	A Swiss diversion: Bernoulli and Euler . . . . .	568
18.5.1	Bernoulli numbers revisited . . . . .	574
18.6	An Italian excursion: Fibonacci numbers . . . . .	575
18.7	A German gem: the Riemann zeta function . . . . .	579
18.7.1	Computing the Riemann zeta function . . . . .	583
18.7.2	Greek relatives of the Riemann zeta function . . . . .	587
18.8	Further reading . . . . .	590
18.9	Summary . . . . .	591
<b>19</b>	<b>Error and probability functions</b> . . . . .	<b>593</b>
19.1	Error functions . . . . .	593
19.1.1	Properties of the error functions . . . . .	593
19.1.2	Computing the error functions . . . . .	595
19.2	Scaled complementary error function . . . . .	598
19.3	Inverse error functions . . . . .	600
19.3.1	Properties of the inverse error functions . . . . .	601
19.3.2	Historical algorithms for the inverse error functions . . . . .	603
19.3.3	Computing the inverse error functions . . . . .	605
19.4	Normal distribution functions and inverses . . . . .	610
19.5	Summary . . . . .	617
<b>20</b>	<b>Elliptic integral functions</b> . . . . .	<b>619</b>
20.1	The arithmetic-geometric mean . . . . .	619
20.2	Elliptic integral functions of the first kind . . . . .	624
20.3	Elliptic integral functions of the second kind . . . . .	627
20.4	Elliptic integral functions of the third kind . . . . .	630
20.5	Computing $K(m)$ and $K'(m)$ . . . . .	631
20.6	Computing $E(m)$ and $E'(m)$ . . . . .	637
20.7	Historical algorithms for elliptic integrals . . . . .	643
20.8	Auxiliary functions for elliptic integrals . . . . .	645
20.9	Computing the elliptic auxiliary functions . . . . .	648
20.10	Historical elliptic functions . . . . .	650
20.11	Elliptic functions in software . . . . .	652
20.12	Applications of elliptic auxiliary functions . . . . .	653
20.13	Elementary functions from elliptic auxiliary functions . . . . .	654
20.14	Computing elementary functions via $R_C(x, y)$ . . . . .	655
20.15	Jacobian elliptic functions . . . . .	657
20.15.1	Properties of Jacobian elliptic functions . . . . .	659
20.15.2	Computing Jacobian elliptic functions . . . . .	661
20.16	Inverses of Jacobian elliptic functions . . . . .	664
20.17	The modulus and the nome . . . . .	668
20.18	Jacobian theta functions . . . . .	673
20.19	Logarithmic derivatives of the Jacobian theta functions . . . . .	675
20.20	Neville theta functions . . . . .	678
20.21	Jacobian Eta, Theta, and Zeta functions . . . . .	679
20.22	Weierstrass elliptic functions . . . . .	682

20.23	Weierstrass functions by duplication . . . . .	689
20.24	Complete elliptic functions, revisited . . . . .	690
20.25	Summary . . . . .	691
<b>21</b>	<b>Bessel functions</b>	<b>693</b>
21.1	Cylindrical Bessel functions . . . . .	694
21.2	Behavior of $J_n(x)$ and $Y_n(x)$ . . . . .	695
21.3	Properties of $J_n(z)$ and $Y_n(z)$ . . . . .	697
21.4	Experiments with recurrences for $J_0(x)$ . . . . .	705
21.5	Computing $J_0(x)$ and $J_1(x)$ . . . . .	707
21.6	Computing $J_n(x)$ . . . . .	710
21.7	Computing $Y_0(x)$ and $Y_1(x)$ . . . . .	713
21.8	Computing $Y_n(x)$ . . . . .	715
21.9	Improving Bessel code near zeros . . . . .	716
21.10	Properties of $I_n(z)$ and $K_n(z)$ . . . . .	718
21.11	Computing $I_0(x)$ and $I_1(x)$ . . . . .	724
21.12	Computing $K_0(x)$ and $K_1(x)$ . . . . .	726
21.13	Computing $I_n(x)$ and $K_n(x)$ . . . . .	728
21.14	Properties of spherical Bessel functions . . . . .	731
21.15	Computing $j_n(x)$ and $y_n(x)$ . . . . .	735
21.16	Improving $j_1(x)$ and $y_1(x)$ . . . . .	740
21.17	Modified spherical Bessel functions . . . . .	743
21.17.1	Computing $i_0(x)$ . . . . .	744
21.17.2	Computing $is_0(x)$ . . . . .	746
21.17.3	Computing $i_1(x)$ . . . . .	747
21.17.4	Computing $is_1(x)$ . . . . .	749
21.17.5	Computing $i_n(x)$ . . . . .	750
21.17.6	Computing $is_n(x)$ . . . . .	753
21.17.7	Computing $k_n(x)$ and $ks_n(x)$ . . . . .	754
21.18	Software for Bessel-function sequences . . . . .	755
21.19	Retrospective on Bessel functions . . . . .	761
<b>22</b>	<b>Testing the library</b>	<b>763</b>
22.1	Testing <code>tgamma()</code> and <code>lgamma()</code> . . . . .	765
22.2	Testing <code>psi()</code> and <code>psiln()</code> . . . . .	768
22.3	Testing <code>erf()</code> and <code>erfc()</code> . . . . .	768
22.4	Testing cylindrical Bessel functions . . . . .	769
22.5	Testing exponent/significand manipulation . . . . .	769
22.6	Testing inline assembly code . . . . .	769
22.7	Testing with Maple . . . . .	770
22.8	Testing floating-point arithmetic . . . . .	773
22.9	The Berkeley Elementary Functions Test Suite . . . . .	774
22.10	The AT&T floating-point test package . . . . .	775
22.11	The Antwerp test suite . . . . .	776
22.12	Summary . . . . .	776
<b>23</b>	<b>Pair-precision elementary functions</b>	<b>777</b>
23.1	Pair-precision integer power . . . . .	777
23.2	Pair-precision machine epsilon . . . . .	779
23.3	Pair-precision exponential . . . . .	780
23.4	Pair-precision logarithm . . . . .	787
23.5	Pair-precision logarithm near one . . . . .	793
23.6	Pair-precision exponential near zero . . . . .	793
23.7	Pair-precision base- $n$ exponentials . . . . .	795
23.8	Pair-precision trigonometric functions . . . . .	796

23.9	Pair-precision inverse trigonometric functions . . . . .	801
23.10	Pair-precision hyperbolic functions . . . . .	804
23.11	Pair-precision inverse hyperbolic functions . . . . .	808
23.12	Summary . . . . .	808
<b>24</b>	<b>Accuracy of the Cody/Waite algorithms</b>	<b>811</b>
<b>25</b>	<b>Improving upon the Cody/Waite algorithms</b>	<b>823</b>
25.1	The Bell Labs libraries . . . . .	823
25.2	The Cephes library . . . . .	823
25.3	The Sun libraries . . . . .	824
25.4	Mathematical functions on EPIC . . . . .	824
25.5	The GNU libraries . . . . .	825
25.6	The French libraries . . . . .	825
25.7	The NIST effort . . . . .	826
25.8	Commercial mathematical libraries . . . . .	826
25.9	Mathematical libraries for decimal arithmetic . . . . .	826
25.10	Mathematical library research publications . . . . .	826
25.11	Books on computing mathematical functions . . . . .	827
25.12	Summary . . . . .	828
<b>26</b>	<b>Floating-point output</b>	<b>829</b>
26.1	Output character string design issues . . . . .	830
26.2	Exact output conversion . . . . .	831
26.3	Hexadecimal floating-point output . . . . .	832
26.3.1	Hexadecimal floating-point output requirements . . . . .	832
26.3.2	Remarks on hexadecimal floating-point output . . . . .	833
26.3.3	Hexadecimal floating-point output-conversion code . . . . .	834
26.3.4	Conversion to uppercase . . . . .	848
26.3.5	Determining rounding direction . . . . .	848
26.4	Octal floating-point output . . . . .	850
26.5	Binary floating-point output . . . . .	851
26.6	Decimal floating-point output . . . . .	851
26.6.1	The decimal-conversion program interface . . . . .	853
26.6.2	Fast powers of ten . . . . .	855
26.6.3	Preliminary scaling . . . . .	856
26.6.4	Support for %g-style conversion . . . . .	857
26.6.5	Buffer sizes . . . . .	858
26.6.6	Special cases . . . . .	858
26.6.7	Scaling and rounding adjustment . . . . .	859
26.6.8	Digit generation . . . . .	860
26.6.9	Completing decimal output conversion . . . . .	861
26.6.10	Computing the minimum desirable precision . . . . .	864
26.6.11	Coding fast powers of ten . . . . .	864
26.7	Accuracy of output conversion . . . . .	865
26.8	Output conversion to a general base . . . . .	865
26.9	Output conversion of Infinity . . . . .	866
26.10	Output conversion of NaN . . . . .	866
26.11	Number-to-string conversion . . . . .	867
26.12	The printf() family . . . . .	867
26.12.1	Dangers of printf() . . . . .	868
26.12.2	Variable argument lists . . . . .	870
26.12.3	Implementing the printf() family . . . . .	871
26.12.4	Output conversion specifiers . . . . .	873
26.13	Summary . . . . .	878

<b>27 Floating-point input</b>	<b>879</b>
27.1 Binary floating-point input . . . . .	879
27.1.1 Sign input conversion . . . . .	885
27.1.2 Prefix string matching . . . . .	886
27.1.3 Infinity input conversion . . . . .	887
27.1.4 NaN input conversion . . . . .	887
27.1.5 Power input conversion . . . . .	889
27.1.6 Floating-point suffix conversion . . . . .	890
27.1.7 Integer suffix conversion . . . . .	892
27.1.8 Input rounding adjustment . . . . .	893
27.2 Octal floating-point input . . . . .	894
27.3 Hexadecimal floating-point input . . . . .	895
27.4 Decimal floating-point input . . . . .	895
27.5 Based-number input . . . . .	899
27.6 General floating-point input . . . . .	900
27.7 The <code>scanf()</code> family . . . . .	901
27.7.1 Implementing the <code>scanf()</code> family . . . . .	902
27.7.2 Whitespace and ordinary characters . . . . .	904
27.7.3 Input conversion specifiers . . . . .	905
27.7.4 Retrospective on the <code>scanf()</code> family . . . . .	909
27.8 Summary . . . . .	910
<b>A Ada interface</b>	<b>911</b>
A.1 Building the Ada interface . . . . .	911
A.2 Programming the Ada interface . . . . .	912
A.3 Using the Ada interface . . . . .	915
<b>B C# interface</b>	<b>917</b>
B.1 C# on the CLI virtual machine . . . . .	917
B.2 Building the C# interface . . . . .	918
B.3 Programming the C# interface . . . . .	920
B.4 Using the C# interface . . . . .	922
<b>C C++ interface</b>	<b>923</b>
C.1 Building the C++ interface . . . . .	923
C.2 Programming the C++ interface . . . . .	924
C.3 Using the C++ interface . . . . .	925
<b>D Decimal arithmetic</b>	<b>927</b>
D.1 Why we need decimal floating-point arithmetic . . . . .	927
D.2 Decimal floating-point arithmetic design issues . . . . .	928
D.3 How decimal and binary arithmetic differ . . . . .	931
D.4 Initialization of decimal floating-point storage . . . . .	935
D.5 The <code>&lt;decimal.h&gt;</code> header file . . . . .	936
D.6 Rounding in decimal arithmetic . . . . .	936
D.7 Exact scaling in decimal arithmetic . . . . .	937
<b>E Errata in the Cody/Waite book</b>	<b>939</b>
<b>F Fortran interface</b>	<b>941</b>
F.1 Building the Fortran interface . . . . .	943
F.2 Programming the Fortran interface . . . . .	944
F.3 Using the Fortran interface . . . . .	945

<b>H</b>	<b>Historical floating-point architectures</b>	<b>947</b>
H.1	CDC family . . . . .	949
H.2	Cray family . . . . .	952
H.3	DEC PDP-10 . . . . .	953
H.4	DEC PDP-11 and VAX . . . . .	956
H.5	General Electric 600 series . . . . .	958
H.6	IBM family . . . . .	959
H.6.1	IBM 7030 Stretch . . . . .	959
H.6.2	IBM and Fortran . . . . .	961
H.6.3	IBM System/360 . . . . .	963
H.7	Lawrence Livermore S-1 Mark IIA . . . . .	965
H.8	Unusual floating-point systems . . . . .	966
H.9	Historical retrospective . . . . .	967
<b>I</b>	<b>Integer arithmetic</b>	<b>969</b>
I.1	Memory addressing and integers . . . . .	971
I.2	Representations of signed integers . . . . .	971
I.2.1	Sign-magnitude representation . . . . .	971
I.2.2	One's-complement representation . . . . .	972
I.2.3	Two's-complement representation . . . . .	972
I.2.4	Excess- <i>n</i> representation . . . . .	974
I.2.5	Ranges of integers . . . . .	974
I.3	Parity testing . . . . .	975
I.4	Sign testing . . . . .	975
I.5	Arithmetic exceptions . . . . .	975
I.6	Notations for binary numbers . . . . .	977
I.7	Summary . . . . .	978
<b>J</b>	<b>Java interface</b>	<b>979</b>
J.1	Building the Java interface . . . . .	979
J.2	Programming the Java MathCW class . . . . .	980
J.3	Programming the Java C interface . . . . .	982
J.4	Using the Java interface . . . . .	985
<b>L</b>	<b>Letter notation</b>	<b>987</b>
<b>P</b>	<b>Pascal interface</b>	<b>989</b>
P.1	Building the Pascal interface . . . . .	989
P.2	Programming the Pascal MathCW module . . . . .	990
P.3	Using the Pascal module interface . . . . .	993
P.4	Pascal and numeric programming . . . . .	994
	<b>Bibliography</b>	<b>995</b>
	<b>Author/editor index</b>	<b>1039</b>
	<b>Function and macro index</b>	<b>1049</b>
	<b>Subject index</b>	<b>1065</b>
	<b>Colophon</b>	<b>1115</b>

# List of figures

2.1	Asymptotic expansion accuracy for $\operatorname{erfc}(x)$ . . . . .	20
3.1	Exponential function approximation, the wrong way . . . . .	26
3.2	Exponential function approximation, the right way . . . . .	27
3.3	Error in minimax $\langle 3/3 \rangle$ fit of $\sin(x)/x$ . . . . .	37
3.4	Chebyshev polynomials . . . . .	46
3.5	Errors in Chebyshev approximations . . . . .	48
4.1	IEEE 754 binary floating-point data layout . . . . .	63
4.2	Binary and decimal ulp spacing . . . . .	81
5.1	IA-32 floating-point control word . . . . .	125
6.1	Computing the remainder . . . . .	145
7.1	The correlation problem in LCGs . . . . .	171
7.2	Frames from animated rotation of LCG 3-D point sequence . . . . .	172
7.3	Frames from animated rotation of LCG 3-D point sequence . . . . .	173
7.4	Views of uniform distributions of random numbers . . . . .	190
7.5	Views of exponential distributions of random numbers . . . . .	191
7.6	Views of logarithmic distributions of random numbers . . . . .	193
7.7	Views of normal distributions of random numbers . . . . .	193
7.8	Normal curves for various $\sigma$ values . . . . .	195
7.9	Counts of heads for 100 coin-flip experiments . . . . .	197
7.10	Counts of heads for 100 000 coin-flip experiments . . . . .	197
10.1	The exponential function . . . . .	268
10.2	Errors in Cody/Waite $\operatorname{EXP}()$ functions . . . . .	272
10.3	Errors in $\operatorname{EXP}()$ functions . . . . .	273
10.4	Errors in $\operatorname{EXPM1}()$ functions . . . . .	278
10.5	The natural logarithm function . . . . .	282
10.6	Errors in mathcw logarithm functions in a binary base . . . . .	288
10.7	Errors in mathcw logarithm functions in a decimal base . . . . .	289
10.8	Errors in $\operatorname{LOG1P}()$ functions . . . . .	293
11.1	Sine and cosine . . . . .	300
11.2	Cosecant and secant . . . . .	300
11.3	Tangent and cotangent . . . . .	302
11.4	Errors in $\operatorname{COS}()$ functions . . . . .	311
11.5	Errors in $\operatorname{SIN}()$ functions . . . . .	312
11.6	Errors in $\operatorname{TAN}()$ functions . . . . .	313
11.7	Inverse cosine and sine . . . . .	324
11.8	Errors in $\operatorname{ACOS}()$ functions . . . . .	326
11.9	Errors in $\operatorname{ASIN}()$ functions . . . . .	327
11.10	Inverse tangent . . . . .	332
11.11	Errors in $\operatorname{ATAN}()$ functions . . . . .	333
12.1	Hyperbolic functions . . . . .	342
12.2	Errors in hyperbolic functions . . . . .	346

12.3	Inverse hyperbolic functions near origin . . . . .	348
12.4	Inverse hyperbolic cosine . . . . .	349
12.5	Inverse hyperbolic sine . . . . .	349
12.6	Inverse hyperbolic tangent . . . . .	350
12.7	Errors in inverse hyperbolic functions . . . . .	351
13.1	Errors in pair-precision square-root functions . . . . .	376
13.2	Errors in pair-precision square-root functions . . . . .	377
13.3	Errors in pair-precision cube-root functions . . . . .	380
13.4	Errors in pair-precision cube-root functions . . . . .	381
15.1	Cartesian and polar forms of point in complex plane . . . . .	444
15.2	Projecting a point onto the Riemann sphere . . . . .	461
16.1	Three cases for roots of quadratic equations . . . . .	466
17.1	Complex square root surfaces . . . . .	481
17.2	Errors in <code>csqrt()</code> function . . . . .	484
17.3	Complex cube root surfaces . . . . .	485
17.4	Errors in <code>ccbrt()</code> function . . . . .	488
17.5	Complex exponential surfaces . . . . .	489
17.6	Errors in <code>cexp()</code> function . . . . .	493
17.7	Errors in <code>cexpm1()</code> function . . . . .	494
17.8	Complex logarithm surfaces . . . . .	495
17.9	Errors in <code>clog()</code> function . . . . .	497
17.10	Errors in <code>clog1p()</code> function . . . . .	501
17.11	Errors in <code>cpow()</code> function . . . . .	502
17.12	Complex cosine surfaces . . . . .	503
17.13	Complex sine surfaces . . . . .	503
17.14	Complex tangent surfaces . . . . .	503
17.15	Errors in <code>ccos()</code> function . . . . .	505
17.16	Errors in <code>csin()</code> function . . . . .	505
17.17	Errors in <code>ctan()</code> function . . . . .	505
17.18	Complex inverse cosine surfaces . . . . .	506
17.19	Complex inverse sine surfaces . . . . .	506
17.20	Complex inverse tangent surfaces . . . . .	506
17.21	Errors in <code>cacos()</code> function . . . . .	510
17.22	Errors in <code>casin()</code> function . . . . .	510
17.23	Errors in <code>catan()</code> function . . . . .	510
17.24	Complex hyperbolic cosine surfaces . . . . .	511
17.25	Complex hyperbolic sine surfaces . . . . .	511
17.26	Complex hyperbolic tangent surfaces . . . . .	511
17.27	Errors in <code>ccosh()</code> function . . . . .	515
17.28	Errors in <code>csinh()</code> function . . . . .	515
17.29	Errors in <code>ctanh()</code> function . . . . .	515
17.30	Complex inverse hyperbolic cosine surfaces . . . . .	516
17.31	Complex inverse hyperbolic sine surfaces . . . . .	516
17.32	Complex inverse hyperbolic tangent surfaces . . . . .	516
17.33	Errors in <code>cacosh()</code> function . . . . .	519
17.34	Errors in <code>casinh()</code> function . . . . .	519
17.35	Errors in <code>catanh()</code> function . . . . .	519
18.1	Gamma function and its logarithm . . . . .	522
18.2	Errors in the <code>TGAMMA()</code> functions (narrow and log range) . . . . .	532
18.3	Errors in the <code>TGAMMA()</code> functions (wide range) . . . . .	533



18.4	Errors in the LGAMMA() functions (narrow and log range)	537
18.5	Errors in the LGAMMA() functions (wide range)	538
18.6	Psi and psiln functions	539
18.7	Errors in the PSI() functions	546
18.8	Errors in the PSILN() functions	547
18.9	Polygamma functions	549
18.10	Errors in the PGAMMA(n, x) functions	559
18.11	Errors in the gamibf() and gamibdf() functions	567
18.12	Errors in the gamib() and gamibd() functions	568
18.13	Pascal's Triangle and Fibonacci numbers	576
18.14	Riemann zeta function	580
18.15	Errors in the ZETA() functions	585
18.16	Errors in the ZETAM1() functions	586
18.17	Catalan/Dirichlet beta function	588
19.1	Error functions and normal curve	594
19.2	Error-magnification factors for error functions	596
19.3	Errors in the ordinary error functions	599
19.4	Errors in the complementary error functions	600
19.5	Errors in the scaled complementary error functions	601
19.6	Inverse error functions	602
19.7	Error-magnification factors for inverse error functions	604
19.8	Inverse error function and two approximations to it	606
19.9	Relative error in approximation to inverse error function	607
19.10	Errors in inverse error functions	611
19.11	Errors in complementary inverse error functions	611
19.12	Normal distribution functions and inverses	614
20.1	Complete elliptic integral functions of first kind	625
20.2	Complete elliptic integral functions of second kind	628
20.3	Error magnification in elliptic integral functions	629
20.4	Errors in elliptic integral functions of first kind	634
20.5	Errors in complementary elliptic integral functions of first kind	637
20.6	Errors in elliptic integral functions of second kind	641
20.7	Errors in complementary elliptic integral functions of second kind	644
20.8	Jacobian elliptic functions	658
20.9	Jacobian elliptic function amplitudes	660
20.10	Errors in Jacobian elliptic functions along $k$	665
20.11	Errors in Jacobian elliptic functions along $u$	666
20.12	Elliptic modulus and nomes	668
20.13	Jacobian theta functions	674
20.14	Neville theta functions	679
20.15	Jacobian Eta, Theta, and Zeta functions	680
20.16	Weierstrass elliptic functions	684
20.17	Weierstrass sigma and zeta functions	687
20.18	Errors in Weierstrass sigma and zeta functions	688
21.1	Bessel functions $J_n(x)$ and $Y_n(x)$	696
21.2	Bessel function ratios	701
21.3	Errors in $j_0(x)$ functions	709
21.4	Errors in $j_1(x)$ functions	710
21.5	Errors in $j_n(n, x)$ functions	713
21.6	Errors in $y_0(x)$ functions	714
21.7	Errors in $y_1(x)$ functions	715
21.8	Errors in $y_n(n, x)$ functions	717

21.9	Errors in improved $j_0(x)$ function	719
21.10	Modified Bessel functions	720
21.11	Errors in $bi_0(x)$ functions	726
21.12	Errors in $bi_1(x)$ functions	727
21.13	Errors in $bin(x)$ functions	728
21.14	Errors in $bis_0(x)$ functions	729
21.15	Errors in $bis_1(x)$ functions	730
21.16	Errors in $bisn(x)$ functions	731
21.17	Errors in $bk_0(x)$ functions	732
21.18	Errors in $bk_1(x)$ functions	733
21.19	Errors in $bkn(x)$ functions	734
21.20	Errors in $bks_0(x)$ functions	735
21.21	Errors in $bks_1(x)$ functions	736
21.22	Errors in $bksn(x)$ functions	737
21.23	Spherical Bessel functions	739
21.24	Errors in $sbj_0(x)$ functions	741
21.25	Errors in $sbj_1(x)$ functions	742
21.26	Errors in $sbjn(x)$ functions	743
21.27	Errors in $sby_0(x)$ functions	744
21.28	Errors in $sby_1(x)$ functions	745
21.29	Errors in $sbyn(x)$ functions	746
21.30	Errors in $sbi_0(x)$ functions	747
21.31	Errors in $sbi_0(x)$ functions	748
21.32	Errors in $sbi_1(x)$ functions	749
21.33	Errors in $sbis_1(x)$ functions	750
21.34	Errors in $sbin(n, x)$ functions	754
21.35	Errors in $sbisn(n, x)$ functions	755
21.36	Errors in $sbk_0(x)$ functions	756
21.37	Errors in $sbk_1(x)$ functions	756
21.38	Errors in $sbkn(x)$ functions	756
21.39	Errors in $sbks_0(x)$ functions	757
21.40	Errors in $sbks_1(x)$ functions	757
21.41	Errors in $sbksn(x)$ functions	757
22.1	Recurrence relations for log-gamma	767
22.2	Significance loss for log-gamma	767
22.3	Reciprocal of gamma function	768
23.1	Errors in pair-precision exponential functions	786
23.2	Errors in the decimal pair-precision $PEXPD()$ functions	786
23.3	Errors in pair-precision logarithm functions (series region)	792
23.4	Errors in pair-precision logarithm functions	793
23.5	Errors in pair-precision logarithm-plus-one functions	794
23.6	Errors in pair-precision $PEXP1()$ functions	795
23.7	Errors in pair-precision $PEXP2()$ functions	796
23.8	Errors in pair-precision $PEXP8()$ functions	797
23.9	Errors in pair-precision $PEXP10()$ functions	798
23.10	Errors in pair-precision $PEXP16()$ functions	799
23.11	Errors in pair-precision $PCOS()$ functions	800
23.12	Errors in pair-precision $PSIN()$ functions	801
23.13	Errors in pair-precision $PTAN()$ functions	802
23.14	Errors in pair-precision $PACOS()$ functions	805
23.15	Errors in pair-precision $PASIN()$ functions	806
23.16	Errors in pair-precision $PATAN()$ functions	806
23.17	Errors in pair-precision $PCOSH()$ functions	807

23.18	Errors in pair-precision PSINH() functions . . . . .	807
23.19	Errors in pair-precision PTANH() functions . . . . .	809
23.20	Errors in pair-precision PACOSH() functions . . . . .	809
23.21	Errors in pair-precision PASINH() functions . . . . .	810
23.22	Errors in pair-precision PATANH() functions . . . . .	810
D.1	IEEE 754-2008 decimal floating-point data layout . . . . .	930
D.2	IEEE 754-2008 decimal floating-point data layout . . . . .	930
H.1	VAX binary floating-point logical data layout . . . . .	957
H.2	VAX F-floating memory layout . . . . .	957
H.3	VAX D- and G-floating memory layout . . . . .	958
H.4	VAX H-floating memory layout . . . . .	958
H.5	System/360 hexadecimal floating-point data layout . . . . .	964

# List of tables

3.1	Accuracy of rational polynomials . . . . .	34
3.2	Low-level polynomial approximations . . . . .	44
3.3	Chebyshev polynomials and recurrence relation . . . . .	45
3.4	Standard contents of the mathcw library . . . . .	58
3.5	Extended contents of the mathcw library, part 1 . . . . .	59
3.6	Extended contents of the mathcw library, part 2 . . . . .	60
4.1	Magnification factors . . . . .	62
4.2	Binary floating-point characteristics and limits . . . . .	65
4.3	Rounding in binary arithmetic . . . . .	67
4.4	IEEE 754 rounding-mode actions . . . . .	68
4.5	Nonstandardness of $\text{atan2}(\pm 0, \pm 0)$ . . . . .	71
5.1	Interval-arithmetic operations . . . . .	115
6.1	Out-of-range conversions to integers . . . . .	131
6.2	Rounding to integer values . . . . .	132
6.3	Fixed-point representation of large integers . . . . .	133
6.4	Remainder function examples . . . . .	144
6.5	Loop counts in $\text{fmod}()$ . . . . .	148
7.1	Percentage points of chi-square distribution . . . . .	198
9.1	Trigonometric argument reduction . . . . .	244
9.2	Worst cases for trigonometric argument reduction . . . . .	252
10.1	Exponential and logarithm hardware instructions . . . . .	294
11.1	Bernoulli numbers of even order . . . . .	304
11.2	Distances of closest machine numbers to $\frac{1}{2}\pi$ . . . . .	305
11.3	Trigonometric hardware instructions . . . . .	338
12.1	PORT library hyperbolic functions . . . . .	352
12.2	FNLIB library hyperbolic functions . . . . .	352
13.1	Pair-precision primitives . . . . .	356
13.2	Splitting numbers into sums of parts . . . . .	360
13.3	Accuracy of <code>float_pair</code> primitives . . . . .	382
13.4	Accuracy of <code>double_pair</code> primitives . . . . .	383
13.5	Error distribution for <code>float_pair</code> primitives . . . . .	384
13.6	Error distribution for <code>double_pair</code> primitives . . . . .	384
14.1	Special cases in the power function . . . . .	413
14.2	More on special cases in the power function . . . . .	414
14.3	Variable limits in the computation of $\log_n(g/a)$ . . . . .	427
14.4	Accuracy of rational polynomial fits for computing $\log_n(g/a)$ . . . . .	428
14.5	Accuracy of rational polynomial fits to $(n^w - 1)/w$ . . . . .	437
14.6	Effect of $q$ on power-function accuracy . . . . .	439
14.7	Effect of $q$ on power-function memory size . . . . .	439

16.1	Solution of the quadratic equation . . . . .	467
17.1	Complex elementary function tree . . . . .	479
17.2	Special cases for complex square root . . . . .	482
17.3	Special cases for complex cube root . . . . .	486
17.4	Special cases for complex exponential . . . . .	490
17.5	Special cases for complex logarithm . . . . .	496
17.6	Special cases for complex inverse trigonometric cosine . . . . .	507
17.7	Special cases for complex hyperbolic cosine . . . . .	513
17.8	Special cases for complex hyperbolic sine . . . . .	513
17.9	Special cases for complex hyperbolic tangent . . . . .	514
17.10	Special cases for complex inverse hyperbolic cosine . . . . .	518
17.11	Special cases for complex inverse hyperbolic sine . . . . .	518
17.12	Special cases for complex inverse hyperbolic tangent . . . . .	518
18.1	Behavior of $\Gamma(x)$ near poles . . . . .	523
18.2	Zeros of $\lgamma(x)$ . . . . .	534
18.3	Zeros of $\psi(x)$ . . . . .	540
18.4	Accuracy of asymptotic series for polygamma functions . . . . .	555
18.5	Euler numbers of even order . . . . .	572
18.6	Tangent numbers of odd order . . . . .	575
18.7	Fibonacci numbers . . . . .	577
19.1	Cumulative distribution function $\Phi(x)$ ( $x \leq 0$ ) . . . . .	612
19.2	Cumulative distribution function $\Phi(x)$ ( $x \geq 0$ ) . . . . .	613
19.3	Probability of exceeding the mean . . . . .	616
20.1	Computing $\pi$ from the AGM . . . . .	622
21.1	Bessel function family . . . . .	695
21.2	Roots of ordinary Bessel functions . . . . .	698
21.3	Asymptotic trigonometric formulas for Bessel functions . . . . .	699
21.4	Series term counts for $J_n(x)$ . . . . .	703
21.5	Iteration counts for continued fraction of $J_n(x)/J_{n-1}(x)$ . . . . .	712
21.6	Low-order spherical Bessel functions . . . . .	738
21.7	Spherical Bessel function limits . . . . .	740
21.8	Convergence of Taylor series of $i_n(x)$ . . . . .	753
24.1	Bit loss on HP/Compaq/DEC Alpha OSF/1 . . . . .	812
24.2	Bit loss on HP/Compaq/DEC Alpha OSF/1 (large MAXTEST) . . . . .	812
24.3	Bit loss on GNU/Linux AMD64 . . . . .	813
24.4	Bit loss on GNU/Linux AMD64 (large MAXTEST) . . . . .	813
24.5	Bit loss on GNU/Linux IA-32 . . . . .	814
24.6	Bit loss on GNU/Linux IA-64 (no FMA) . . . . .	814
24.7	Bit loss on GNU/Linux IA-64 (FMA) . . . . .	815
24.8	Bit loss on HP-UX IA-64 . . . . .	815
24.9	Bit loss on HP-UX PA-RISC . . . . .	816
24.10	Bit loss on IBM AIX on POWER . . . . .	816
24.11	Bit loss on GNU/Linux MIPS R4400SC . . . . .	817
24.12	Bit loss on SGI IRIX MIPS R10000 . . . . .	817
24.13	Bit loss on Mac OS X PowerPC . . . . .	818
24.14	Bit loss on Solaris SPARC . . . . .	818
24.15	Bit loss on Solaris IA-32 . . . . .	819
24.16	Bit loss on GNU/Linux IA-64 for native math library . . . . .	819
24.17	Bit loss on Solaris SPARC for native Sun math library . . . . .	820

24.18	Bit loss on Solaris IA-32 for native Sun math library . . . . .	820
24.19	Bit loss on Solaris SPARC for IBM APMathLib . . . . .	820
24.20	Bit loss on Solaris SPARC for Sun fdlibm . . . . .	821
24.21	Bit loss on Solaris SPARC for Sun libmcr . . . . .	821
24.22	Bit loss on Solaris SPARC for Moshier's Cephes library . . . . .	821
24.23	Bit loss on Solaris IA-32 for Moshier's Cephes library . . . . .	822
26.1	Symbolic flags for formatted output . . . . .	840
26.2	Hard cases for binary to decimal conversion . . . . .	852
26.3	Goldberg/Matula base-conversion precision . . . . .	853
26.4	More symbolic flags for formatted output . . . . .	854
26.5	Output conversion specifiers (part 1) . . . . .	873
26.6	Output conversion specifiers (part 2) . . . . .	874
26.7	Flag-character format modifiers . . . . .	875
26.8	Precision format modifiers . . . . .	876
26.9	Minimum field-width format modifiers . . . . .	876
26.10	Exponent-width format modifiers . . . . .	876
26.11	Digit-grouping format modifiers . . . . .	876
26.12	Number-base format modifiers . . . . .	876
26.13	Output data-length format modifiers . . . . .	877
26.14	Binary and octal floating-point output . . . . .	877
27.1	Input conversion specifiers . . . . .	906
27.2	Input data type format modifiers . . . . .	907
D.1	Decimal floating-point features (DPD encoding) . . . . .	929
D.2	Decimal floating-point features (BID encoding) . . . . .	929
D.3	Behavior of the <code>quantize()</code> function . . . . .	932
D.4	Behavior of the <code>samequantum()</code> function . . . . .	933
D.5	Behavior of the <code>decimal_normalize()</code> function . . . . .	934
D.6	The <code>&lt;decfloat.h&gt;</code> header file . . . . .	937
H.1	Arithmetic of current and historical computer systems . . . . .	948
I.1	Integer word sizes . . . . .	970
I.2	Sign-magnitude integers . . . . .	971
I.3	One's-complement integers . . . . .	972
I.4	Two's-complement integers . . . . .	973
I.5	Excess- <i>n</i> 4-bit integers . . . . .	974
I.6	Largest integers in various word sizes . . . . .	975
I.7	Behavior of integer division by zero . . . . .	977
L.1	Latin letters in mathematics . . . . .	987
L.2	Greek letters in mathematics . . . . .	988
P.1	Pascal numeric data types . . . . .	990

# Quick start

THIS QUICK START FEATURE IS NOT COMPREHENSIVE:  
THERE MAY BE ADDITIONAL OSHA STANDARDS AND  
GUIDANCE MATERIALS THAT ALSO APPLY.

— WEB SITE DISCLAIMER, U.S. DEPARTMENT OF LABOR,  
OCCUPATIONAL SAFETY & HEALTH ADMINISTRATION.

The `mathcw` library implements the elementary mathematical functions mandated by C89 and C99, normally supplied by the `-lm` library on most UNIX systems. [Table 3.4](#) on page 58 summarizes the `mathcw` library contents.

Much more information is provided in the remaining chapters of this book, but if you are only interested in library installation, then this short section provides enough information for that task.

To build, validate, and install this library on almost any modern UNIX or POSIX-compatible system, this widely used GNU recipe does the job:

```
% ./configure && make all check install
```

On Hewlett-Packard HP-UX on IA-64 (Intel Architecture 64-bit), change the target `all` to `all-hp` to get library support for two additional precisions available on that platform; see [Section 4.26](#) on page 100 for details. The corresponding `check-hp` target tests all five supported precisions.

To change the default installation tree from the GNU-standard default of `/usr/local`, define `prefix` to an equivalent value at install time:

```
% make prefix=/depot install
```

That would create the library file `/depot/lib/libmcw.a`.

The `mathcw` library can be used like this with either C or C++ compilers:

```
% cc [ flags ] [ -I$prefix/include ] file(s) [ -L$prefix ] -lmcw
```

The `-I` option is needed if the code includes this package's `mathcw.h` header file. That may not be necessary, if `<math.h>` is included. Depending on the local conventions, the `-L` option may or may not be required to define a load-library path. `$prefix` must be replaced by the local default, such as `/usr/local`.

Caution is needed on systems for which the C header file `<math.h>` redefines library function names, or where compilers produce inline code for elementary functions, such as on the Intel IA-32 (formerly, x86) architecture, which has single hardware instructions for several of them. For the GNU compiler family, use the option `-fno-builtin` to permit library routines to replace inline code.

Caution is also needed when the host default floating-point behavior is not IEEE-754 conformant. GNU/LINUX and OSF/1 operating systems with Alpha processors have this defect: underflows flush abruptly to zero, and overflows and operations with NaN immediately terminate the process. To get mostly conformant behavior, with GNU compilers, use the `-mieee` flag, and with native compilers, use the `-ieee` flag. To get full conformance, use `-mieee-with-inexact` or `-mieee-conformant`, plus `-mfp-rounding-mode=d`, with GNU compilers, and `-ieee_with_inexact` and `-mfp-rounding-mode=d` with native ones.

On IBM AIX 4.2, `long double` is compiled as `double`, unless the native compiler name is changed from `cc` to `cc128`, or the `-qlongdouble` option is specified. Run-time library support of 128-bit `long double` requires linking with `cc128`, or explicit specification of the `-lc128` library. The easiest way to build the `mathcw` library is then like this:

```
% make CC=cc128 all check
```

The GNU compiler `gcc` version 2.95.3 does not support a 128-bit `long double` at all, and no newer version builds successfully on AIX 4.2.

It is possible to compile a subset of the package, either manually to select replacements for deficient implementations in the native `-lm` library, or by precision, with the targets `float`, `double`, and `longdouble`. For example, pre-C99

implementations of the C language may lack some or all of the float and long double elementary functions, providing only the double versions.

To clean up after a build, do this:

```
% make clean
```

To return to the original state of a freshly unpacked distribution, use the command

```
% make distclean
```

More details about the interpretation of the test results are given later in **Chapter 22** on page 763.

The mathcw library distribution includes interfaces that make the library accessible from several other major programming languages. The interfaces are described in appendices of this book, beginning on page 911.

For up-to-date information about building the library, run the command

```
% make help
```

to get further information about platform-specific targets, and about building shared libraries.