Important Note on New Product Names

As part of Sun’s new developer product strategy, we have changed the names of our development tools from Sun WorkShop™ to Forte™ Developer products. The products, as you can see, are the same high-quality products you have come to expect from Sun; the only thing that has changed is the name.

We believe that the Forte™ name blends the traditional quality and focus of Sun’s core programming tools with the multi-platform, business application deployment focus of the Forte tools, such as Forte Fusion™ and Forte™ for Java™. The new Forte organization delivers a complete array of tools for end-to-end application development and deployment.

For users of the Sun WorkShop tools, the following is a simple mapping of the old product names in WorkShop 5.0 to the new names in Forte Developer 6.

<table>
<thead>
<tr>
<th>Old Product Name</th>
<th>New Product Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun Visual WorkShop™ C++</td>
<td>Forte™ C++ Enterprise Edition 6</td>
</tr>
<tr>
<td>Sun Performance WorkShop™ Fortran</td>
<td>Forte™ for High Performance Computing 6</td>
</tr>
<tr>
<td>Sun Performance WorkShop™ Fortran Personal Edition</td>
<td>Forte™ Fortran Desktop Edition 6</td>
</tr>
<tr>
<td>Sun WorkShop Professional™ C</td>
<td>Forte™ C 6</td>
</tr>
<tr>
<td>Sun WorkShop™ University Edition</td>
<td>Forte™ Developer University Edition 6</td>
</tr>
</tbody>
</table>

In addition to the name changes, there have been major changes to two of the products.

- Forte for High Performance Computing contains all the tools formerly found in Sun Performance WorkShop Fortran and now includes the C++ compiler, so High Performance Computing users need to purchase only one product for all their development needs.

- Forte Fortran Desktop Edition is identical to the former Sun Performance WorkShop Personal Edition, except that the Fortran compilers in that product no longer support the creation of automatically parallelized or explicit, directive-based parallel code. This capability is still supported in the Fortran compilers in Forte for High Performance Computing.

We appreciate your continued use of our development products and hope that we can continue to fulfill your needs into the future.
Contents

Preface   P-1

1. Introduction to C++ Libraries   1-1
   1.1 Man Pages   1-1
   1.2 Other Libraries   1-2
       1.2.1 Tools.h++ Library   1-2
       1.2.2 Sun WorkShop Memory Monitor   1-2

2. The Complex Arithmetic Library   2-1
   2.1 The Complex Library   2-1
       2.1.1 Using the Complex Library   2-2
   2.2 Type complex   2-2
       2.2.1 Constructors of Class complex   2-2
       2.2.2 Arithmetic Operators   2-3
   2.3 Mathematical Functions   2-4
   2.4 Error Handling   2-6
   2.5 Input and Output   2-7
   2.6 Mixed-Mode Arithmetic   2-8
   2.7 Efficiency   2-9
   2.8 Complex Man Pages   2-10
3. **The Classic iostream Library** 3-1

3.1 Predefined iostreams 3-1
3.2 Basic Structure of iostream Interaction 3-2
3.3 Using the Classic iostream Library 3-3
  3.3.1 Output Using iostream 3-4
  3.3.2 Input Using iostream 3-7
  3.3.3 Defining Your Own Extraction Operators 3-7
  3.3.4 Using the char* Extractor 3-8
  3.3.5 Reading Any Single Character 3-9
  3.3.6 Binary Input 3-9
  3.3.7 Peeking at Input 3-9
  3.3.8 Extracting Whitespace 3-10
  3.3.9 Handling Input Errors 3-10
  3.3.10 Using iostreams with stdio 3-11
3.4 Creating iostreams 3-11
  3.4.1 Dealing with Files Using Class fstream 3-11
3.5 Assignment of iostreams 3-15
3.6 Format Control 3-15
3.7 Manipulators 3-15
  3.7.1 Using Plain Manipulators 3-17
  3.7.2 Parameterized Manipulators 3-18
3.8 Strstreams: iostreams for Arrays 3-20
3.9 Stdiobufs: iostreams for stdio Files 3-20
3.10 Streambufs 3-20
  3.10.1 Working with Streambufs 3-20
  3.10.2 Using Streambufs 3-21
3.11 iostream Man Pages 3-22
3.12 iostream Terminology 3-24
4. Using Classic *iostreams* in a Multithreading Environment  4-1
   4.1 Multithreading  4-1
   4.2 Organization of the MT-Safe *iostream* Library  4-2
      4.2.1 Public Conversion Routines  4-3
      4.2.2 Compiling and Linking with the MT-Safe *libC* Library  4-4
      4.2.3 MT-Safe *iostream* Restrictions  4-5
      4.2.4 Performance  4-7
   4.3 Interface Changes to the *iostream* Library  4-9
      4.3.1 New Classes  4-9
      4.3.2 New Class Hierarchy  4-10
      4.3.3 New Functions  4-10
   4.4 Global and Static Data  4-12
   4.5 Sequence Execution  4-13
   4.6 Object Locks  4-13
      4.6.1 Class *stream_locker*  4-14
   4.7 MT-Safe Classes  4-15
   4.8 Object Destruction  4-16
   4.9 An Example Application  4-17

5. The C++ Standard Library  5-1
   5.1 C++ Standard Library Header Files  5-2
   5.2 C++ Standard Library Man Pages  5-3

Index  Index-1
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE P-1</td>
<td>Typographic Conventions</td>
<td>P-3</td>
</tr>
<tr>
<td>TABLE P-2</td>
<td>Shell Prompts</td>
<td>P-4</td>
</tr>
<tr>
<td>TABLE P-3</td>
<td>Related Sun WorkShop 6 Documentation by Document Collection</td>
<td>P-5</td>
</tr>
<tr>
<td>TABLE P-4</td>
<td>Related Solaris Documentation</td>
<td>P-8</td>
</tr>
<tr>
<td>TABLE P-5</td>
<td>Man Pages Related to C++</td>
<td>P-9</td>
</tr>
<tr>
<td>TABLE 1-1</td>
<td>Man Pages for Sun WorkShop Memory Monitor</td>
<td>1-3</td>
</tr>
<tr>
<td>TABLE 2-1</td>
<td>Complex Arithmetic Library Functions</td>
<td>2-4</td>
</tr>
<tr>
<td>TABLE 2-2</td>
<td>Complex Mathematical and Trigonometric Functions</td>
<td>2-5</td>
</tr>
<tr>
<td>TABLE 2-3</td>
<td>Complex Arithmetic Library Functions</td>
<td>2-7</td>
</tr>
<tr>
<td>TABLE 2-4</td>
<td>Man Pages for Type complex</td>
<td>2-10</td>
</tr>
<tr>
<td>TABLE 3-1</td>
<td>iostream Routine Header Files</td>
<td>3-3</td>
</tr>
<tr>
<td>TABLE 3-2</td>
<td>iostream Predefined Manipulators</td>
<td>3-16</td>
</tr>
<tr>
<td>TABLE 3-3</td>
<td>iostream Man Pages Overview</td>
<td>3-22</td>
</tr>
<tr>
<td>TABLE 3-4</td>
<td>iostream Terminology</td>
<td>3-24</td>
</tr>
<tr>
<td>TABLE 4-1</td>
<td>Core Classes</td>
<td>4-2</td>
</tr>
<tr>
<td>TABLE 4-2</td>
<td>Reentrant Public Functions</td>
<td>4-3</td>
</tr>
<tr>
<td>TABLE 5-1</td>
<td>C++ Standard Library Header Files</td>
<td>5-2</td>
</tr>
<tr>
<td>TABLE 5-2</td>
<td>Man Pages for C++ Standard Library</td>
<td>5-3</td>
</tr>
</tbody>
</table>
Preface

The C++ Library Reference describes the C++ libraries, including:
- Tools.h++ Class Library
- Sun WorkShop Memory Monitor
- Complex

Multiplatform Release

This Sun WorkShop release supports versions 2.6, 7, and 8 of the Solaris™ SPARC™

Note – In this document, the term “IA” refers to the Intel 32-bit processor
architecture, which includes the Pentium, Pentium Pro, and Pentium II, Pentium II
Xeon, Celeron, Pentium III, and Pentium III Xeon processors and compatible
microprocessor chips made by AMD and Cyrix.

Access to Sun WorkShop Development Tools

Because Sun WorkShop product components and man pages do not install into the
standard /usr/bin/ and /usr/share/man directories, you must change your
PATH and MANPATH environment variables to enable access to Sun WorkShop
compilers and tools.
To determine if you need to set your PATH environment variable:

1. **Display the current value of the PATH variable by typing:**
   
   ```
   % echo $PATH
   ```

2. **Review the output for a string of paths containing /opt/SUNWspro/bin/**.

   If you find the paths, your PATH variable is already set to access Sun WorkShop development tools. If you do not find the paths, set your PATH environment variable by following the instructions in this section.

To determine if you need to set your MANPATH environment variable:

1. **Request the workshop man page by typing:**
   
   ```
   % man workshop
   ```

2. **Review the output, if any.**

   If the workshop(1) man page cannot be found or if the man page displayed is not for the current version of the software installed, follow the instructions in this section for setting your MANPATH environment variable.

---

**Note** – The information in this section assumes that your Sun WorkShop 6 products were installed in the /opt directory. If your Sun WorkShop software is not installed in the /opt directory, ask your system administrator for the equivalent path on your system.

---

The PATH and MANPATH variables should be set in your home .cshrc file if you are using the C shell or in your home .profile file if you are using the Bourne or Korn shells:

- To use Sun WorkShop commands, add the following to your PATH variable:
  
  ```
  /opt/SUNWspro/bin
  ```

- To access Sun WorkShop man pages with the man command, add the following to your MANPATH variable:
  
  ```
  /opt/SUNWspro/man
  ```

For more information about the PATH variable, see the csh(1), sh(1), and ksh(1) man pages. For more information about the MANPATH variable, see the man(1) man page. For more information about setting your PATH and MANPATH variables to access this release, see the *Sun WorkShop 6 Installation Guide* or your system administrator.
How This Book Is Organized

This book contains the following chapters:

Chapter 1, “Introduction to C++ Libraries,” gives an overview of the C++ libraries.

Chapter 2, “The Complex Arithmetic Library,” explains the arithmetic operators and mathematical functions in the library.

Chapter 3, “The Classic iostream Library,” discusses the classic implementation of the input and output facility used in C++.

Chapter 4, “Using Classic iostreams in a Multithreading Environment,” details how to use the classic iostream library for input and output in a multithreaded environment.


Typographic Conventions

TABLE P-1 shows the typographic conventions that are used in Sun WorkShop documentation.

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories; on-screen computer output</td>
<td>Edit your .login file. Use ls -a to list all files. % You have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, when contrasted with on-screen computer output</td>
<td>% su Password:</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new words or terms, words to be emphasized</td>
<td>Read Chapter 6 in the User’s Guide. These are called class options. You must be superuser to do this.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Command-line placeholder text; replace with a real name or value</td>
<td>To delete a file, type rm filename.</td>
</tr>
</tbody>
</table>
Shell Prompts

TABLE P-2 shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

<table>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell</td>
<td>%</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>$</td>
</tr>
<tr>
<td>C shell, Bourne shell, and Korn shell superuser</td>
<td>#</td>
</tr>
</tbody>
</table>

Related Documentation

You can access documentation related to the subject matter of this book in the following ways:
Through the Internet at the docs.sun.com\textsuperscript{sm} Web site. You can search for a specific book title or you can browse by subject, document collection, or product at the following Web site:

http://docs.sun.com

Through the installed Sun WorkShop products on your local system or network. Sun WorkShop 6 HTML documents (manuals, online help, man pages, component readme files, and release notes) are available with your installed Sun WorkShop 6 products. To access the HTML documentation, do one of the following:

- In any Sun WorkShop or Sun WorkShop\textsuperscript{TM} TeamWare window, choose Help ➤ About Documentation.

- In your Netscape\textsuperscript{TM} Communicator 4.0 or compatible version browser, open the following file:

  /opt/SUNWspro/docs/index.html

(If your Sun WorkShop software is not installed in the /opt directory, ask your system administrator for the equivalent path on your system.) Your browser displays an index of Sun WorkShop 6 HTML documents. To open a document in the index, click the document’s title.
Document Collections

TABLE P-3 lists related Sun WorkShop 6 manuals by document collection.

<table>
<thead>
<tr>
<th>Document Collection</th>
<th>Document Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forte™ Developer 6 / Sun WorkShop 6</td>
<td>About Sun WorkShop 6 Documentation</td>
<td>Describes the documentation available with this Sun WorkShop release and how to access it.</td>
</tr>
<tr>
<td></td>
<td>What's New in Sun WorkShop 6</td>
<td>Provides information about the new features in the current and previous release of Sun WorkShop.</td>
</tr>
<tr>
<td></td>
<td>Sun WorkShop 6 Release Notes</td>
<td>Contains installation details and other information that was not available until immediately before the final release of Sun WorkShop 6. This document complements the information that is available in the component readme files.</td>
</tr>
<tr>
<td>Forte Developer 6 / Sun WorkShop 6</td>
<td>Analyzing Program Performance With Sun WorkShop 6</td>
<td>Explains how to use the new Sampling Collector and Sampling Analyzer (with examples and a discussion of advanced profiling topics) and includes information about the command-line analysis tool er_print, the LoopTool and LoopReport utilities, and UNIX profiling tools prof, gprof, and tcov.</td>
</tr>
<tr>
<td></td>
<td>Debugging a Program With dbx</td>
<td>Provides information on using dbx commands to debug a program with references to how the same debugging operations can be performed using the Sun WorkShop Debugging window.</td>
</tr>
<tr>
<td></td>
<td>Introduction to Sun WorkShop</td>
<td>Acquaints you with the basic program development features of the Sun WorkShop integrated programming environment.</td>
</tr>
<tr>
<td>Document Collection</td>
<td>Document Title</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Forte™ C 6 / Sun WorkShop 6 Compilers C</td>
<td>C User's Guide</td>
<td>Describes the C compiler options, Sun-specific capabilities such as pragmas, the lint tool, parallelization, migration to a 64-bit operating system, and ANSI/ISO-compliant C.</td>
</tr>
<tr>
<td>Forte™ C++ 6 / Sun WorkShop 6 Compilers C++</td>
<td>C++ Library Reference</td>
<td>Describes the C++ libraries, including C++ Standard Library, Tools.h++ class library, Sun WorkShop Memory Monitor, Iostream, and Complex.</td>
</tr>
<tr>
<td></td>
<td>C++ Migration Guide</td>
<td>Provides guidance on migrating code to this version of the Sun WorkShop C++ compiler.</td>
</tr>
<tr>
<td></td>
<td>C++ Programming Guide</td>
<td>Explains how to use the new features to write more efficient programs and covers templates, exception handling, runtime type identification, cast operations, performance, and multithreaded programs.</td>
</tr>
<tr>
<td></td>
<td>C++ User's Guide</td>
<td>Provides information on command-line options and how to use the compiler.</td>
</tr>
<tr>
<td></td>
<td>Sun WorkShop Memory Monitor User's Manual</td>
<td>Describes how the Sun WorkShop Memory Monitor solves the problems of memory management in C and C++. This manual is only available through your installed product (see /opt/SUNWspro/docs/index.html) and not at the docs.sun.com Web site.</td>
</tr>
<tr>
<td>Forte™ for High Performance Computing 6 / Sun WorkShop 6 Compilers Fortran 77/95</td>
<td>Fortran Library Reference</td>
<td>Provides details about the library routines supplied with the Fortran compiler.</td>
</tr>
<tr>
<td>Document Collection</td>
<td>Document Title</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Fortran Programming Guide</strong></td>
<td>Discusses issues relating to input/output, libraries, program analysis, debugging, and performance.</td>
<td></td>
</tr>
<tr>
<td><strong>Fortran User’s Guide</strong></td>
<td>Provides information on command-line options and how to use the compilers.</td>
<td></td>
</tr>
<tr>
<td><strong>FORTRAN 77 Language Reference</strong></td>
<td>Provides a complete language reference.</td>
<td></td>
</tr>
<tr>
<td><strong>Interval Arithmetic Programming Reference</strong></td>
<td>Describes the intrinsic INTERVAL data type supported by the Fortran 95 compiler.</td>
<td></td>
</tr>
<tr>
<td><strong>Forte™ TeamWare 6 / Sun WorkShop TeamWare 6 User’s Guide</strong></td>
<td>Describes how to use the Sun WorkShop TeamWare code management tools.</td>
<td></td>
</tr>
<tr>
<td><strong>Forte™ / Sun Performance Library 6</strong></td>
<td>Discusses the optimized library of subroutines and functions used to perform computational linear algebra and fast Fourier transforms.</td>
<td></td>
</tr>
<tr>
<td><strong>Sun Performance Library User’s Guide</strong></td>
<td>Describes how to use the Sun-specific features of the Sun Performance Library, which is a collection of subroutines and functions used to solve linear algebra problems.</td>
<td></td>
</tr>
<tr>
<td><strong>Numerical Computation Guide</strong></td>
<td>Describes issues regarding the numerical accuracy of floating-point computations.</td>
<td></td>
</tr>
<tr>
<td><strong>Standard Library 2</strong></td>
<td>Provides details on the Standard C++ Library.</td>
<td></td>
</tr>
<tr>
<td><strong>Tools.h++ 7</strong></td>
<td>Provides details on the Tools.h++ class library.</td>
<td></td>
</tr>
<tr>
<td><strong>Tools.h++ User’s Guide</strong></td>
<td>Discusses use of the C++ classes for enhancing the efficiency of your programs.</td>
<td></td>
</tr>
</tbody>
</table>
TABLE P-4 describes related Solaris documentation available through the docs.sun.com Web site.

TABLE P-4 Related Solaris Documentation

<table>
<thead>
<tr>
<th>Document Collection</th>
<th>Document Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris Software Developer</td>
<td>Linker and Libraries Guide</td>
<td>Describes the operations of the Solaris link-editor and runtime linker and the objects on which they operate.</td>
</tr>
<tr>
<td></td>
<td>Programming Utilities Guide</td>
<td>Provides information for developers about the special built-in programming tools that are available in the Solaris operating environment.</td>
</tr>
</tbody>
</table>

Man Pages

The C++ Library Reference lists the man pages that are available for the C++ libraries. TABLE P-5 lists other man pages that are related to C++.

TABLE P-5 Man Pages Related to C++

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c++filt</td>
<td>Copies each file name in sequence and writes it in the standard output after decoding symbols that look like C++ demangled names.</td>
</tr>
<tr>
<td>dem</td>
<td>Demangles one or more C++ names that you specify</td>
</tr>
<tr>
<td>fbe</td>
<td>Creates object files from assembly language source files.</td>
</tr>
<tr>
<td>fversion</td>
<td>Prints information about the system CPU and FPU</td>
</tr>
<tr>
<td>gprof</td>
<td>Produces execution profile of a program</td>
</tr>
<tr>
<td>ild</td>
<td>Links incrementally, allowing insertion of modified object code into a previously built executable</td>
</tr>
<tr>
<td>inline</td>
<td>Expands assembler inline procedure calls</td>
</tr>
<tr>
<td>lex</td>
<td>Generates lexical analysis programs</td>
</tr>
<tr>
<td>rpcgen</td>
<td>Generates C/C++ code to implement an RPC protocol</td>
</tr>
<tr>
<td>sigfpe</td>
<td>Allows signal handling for specific SIGFPE codes</td>
</tr>
<tr>
<td>stdarg</td>
<td>Handles variable argument list</td>
</tr>
</tbody>
</table>
TABLE P-5  Man Pages Related to C++  (Continued)

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varargs</td>
<td>Handles variable argument list</td>
</tr>
<tr>
<td>version</td>
<td>Displays version identification of object file or binary</td>
</tr>
<tr>
<td>yacc</td>
<td>Converts a context-free grammar into a set of tables for a simple automaton that executes an LALR(1) parsing algorithm</td>
</tr>
</tbody>
</table>

**README File**

The README file highlights important information about the compiler, including:

- New and changed features
- Software incompatibilities
- Current software bugs
- Information discovered after the manuals were printed

To view the text version of the C++ compiler README file, type the following at a command prompt:

```
example% CC -xhelp=readme
```

To access the HTML version of the README, in your Netscape Communicator 4.0 or compatible version browser, open the following file:

```
/opt/SUNWspro/docs/index.html
```

(If your Sun WorkShop software is not installed in the /opt directory, ask your system administrator for the equivalent path on your system.) Your browser displays an index of Sun WorkShop 6 HTML documents. To open the README, find its entry in the index, then click the title.

---

**Commercially Available Books**

The following is a partial list of available books on the C++ language.


Standard C++ I/OStreams and Locales, Angelika Langer and Klaus Kreft (Addison-Wesley, 2000).


Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, (Addison-Wesley, 1995).


More Effective C++—35 Ways to Improve Your Programs and Designs, Scott Meyers (Addison-Wesley, 1996).
Introduction to C++ Libraries

The C++ class libraries are modular components of reusable code. Using class libraries, you can integrate blocks of code that have been previously built and tested.

A C++ library consists of one or more header files and an object library. The header files provide class and other definitions needed to access the library functions. The object library provides compiled functions and data that are linked with your program to produce an executable program.

This manual describes three class libraries provided with the C++ compiler:

- Complex numbers, described in Chapter 2
- iostreams, described in Chapter 3
- C++ standard library, described in Chapter 5

For a discussion of building shared and static libraries, and using libraries, see the C++ User’s Guide.

1.1 Man Pages

The man pages associated with the libraries described in this manual are located in:

```
install-directory/SUNWspro/man/man3
install-directory/SUNWspro/man/man3cc4
install-directory/SUNWspro/man/man3c++
```

where the default install-directory is /opt.

To access these man pages, ensure that your MANPATH includes install-directory/SUNWspro/man. For instructions on setting your MANPATH, see “Access to Sun WorkShop Development Tools” in the preface.
To access man pages for the Sun WorkShop Compilers C++ libraries, type:

```
example% man library-name
```

To access man pages for the compatibility mode libraries of the Sun WorkShop C++ compiler, type:

```
example% man -s 3CC4 library-name
```

You can also access the man pages by pointing your browser to:

```
file:install-directory/SUNWspro/docs/index.html
```

where the default install-directory is /opt.

---

### 1.2 Other Libraries

In addition to the complex, iostreams, and C++ standard libraries, this release provides the Tools.h++ library and the Sun WorkShop Memory Monitor library.

#### 1.2.1 Tools.h++ Library

Tools.h++ is a C++ foundation class library from RogueWave. Version 7 of this library is provided with this release. For further information about this library, see:

- C++ Users Guide
- Tools.h++ User’s Guide (Version 7)
- Tools.h++ Class Library Reference (Version 7)

#### 1.2.2 Sun WorkShop Memory Monitor

The Sun WorkShop Memory Monitor provides facilities to automatically report and fix memory leaks, memory fragmentation, and premature frees. It has three modes of operation:

- Debugging
- Deployment
Garbage collection

These modes are dependent on the library you link your application with.

The components of the Sun WorkShop Memory Monitor are:

- `libgc`—the library used in garbage collection and deployment modes
- `libgc_dbg`—the library used in memory debugging mode
- `gcmonitor`—the daemon used in memory debugging mode

For complete documentation on the Sun WorkShop Memory Monitor, launch the Memory Monitor or point your browser at the following file:

```
file:install-directory/SUNWspro/docs/index.html
```

Replace `install-directory` with the path to your Sun WorkShop installation directory. In a default installation, `install-directory` is `/opt`.

Man pages for the Sun WorkShop Memory Monitor are located in:

- `install-directory/SUNWspro/man/man1`
- `install-directory/SUNWspro/man/man3`

where the default `install-directory` is `/opt`.

**TABLE 1-1** Man Pages for Sun WorkShop Memory Monitor

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcmonitor</td>
<td>Web interface for Sun WorkShop Memory Monitor</td>
</tr>
<tr>
<td>gcFixPrematureFrees</td>
<td>Enable and disable fixing of premature frees by the Sun WorkShop Memory Monitor</td>
</tr>
<tr>
<td>gcInitialize</td>
<td>Configure Sun WorkShop Memory Monitor at startup</td>
</tr>
</tbody>
</table>
CHAPTER 2

The Complex Arithmetic Library

Complex numbers are numbers made up of a real and an imaginary part. For example:

| 3.2 + 4i  |
| 1 + 3i    |
| 1 + 2.3i  |

In the degenerate case, 0 + 3i is an entirely imaginary number generally written as 3i, and 5 + 0i is an entirely real number generally written as 5. You can represent complex numbers using the complex data type.

Note – The complex arithmetic library (libcomplex) is available only for compatibility mode (-compat=4). In standard mode (the default mode), complex number classes with similar functionality are included with the C++ Standard Library libCstd.

2.1 The Complex Library

The complex arithmetic library implements a complex number data type as a new data type and provides:

- Operators
- Mathematical functions (defined for the built-in numerical types)
- Extensions (for iostreams that allow input and output of complex numbers)
- Error handling mechanisms
Complex numbers can also be represented as an absolute value (or magnitude) and an argument (or angle). The library provides functions to convert between the real and imaginary (Cartesian) representation and the magnitude and angle (polar) representation.

The complex conjugate of a number has the opposite sign in its imaginary part.

2.1.1 Using the Complex Library

To use the complex library, include the header file complex.h in your program, and compile and link with the -library=complex option.

2.2 Type complex

The complex arithmetic library defines one class: class complex. An object of class complex can hold a single complex number. The complex number is constructed of two parts:

- The real part
- The imaginary part

```cpp
class complex {
    double re, im;
};
```

The numerical values of each part are held in fields of type double. Here is the relevant part of the definition of complex:

The value of an object of class complex is a pair of double values. The first value represents the real part; the second value represents the imaginary part.

2.2.1 Constructors of Class complex

There are two constructors for complex. Their definitions are:

```cpp
complex::complex(){ re=0.0; im=0.0; }  
complex::complex(double r, double i = 0.0) { re=r; im=i; }
```
If you declare a complex variable without parameters, the first constructor is used and the variable is initialized, so that both parts are 0. The following example creates a complex variable whose real and imaginary parts are both 0:

```c
complex aComp;
```

You can give either one or two parameters. In either case, the second constructor is used. When you give only one parameter, it is taken as the value for the real part and the imaginary part is set to 0. For example:

```c
complex aComp(4.533);
```

creates a complex variable with the value:

4.533 + 0i

If you give two values, the first is taken as the value of the real part and the second as the value of the imaginary part. For example:

```c
complex aComp(8.999, 2.333);
```

creates a complex variable with the value:

8.999 + 2.333i

You can also create a complex number using the `polar` function, which is provided in the complex arithmetic library (see Section 2.3 “Mathematical Functions”). The `polar` function creates a complex value given a pair of polar coordinates, magnitude and angle.

There is no destructor for type `complex`.

### 2.2.2 Arithmetic Operators

The complex arithmetic library defines all the basic arithmetic operators. Specifically, the following operators work in the usual way and with the usual precedence:

+  -  /  *  =
The operator `-` has its usual binary and unary meanings.

In addition, you can use the following operators in the usual way:

```
+= -= *= /=
```

However, these last four operators do not produce values that you can use in expressions. For example, the following does not work:

```
complex a, b;
...
if ((a+=2)==0) {...}; // illegal
b = a *= b; // illegal
```

You can also use the following equality operators in their regular meaning:

```
== !=
```

When you mix real and complex numbers in an arithmetic expression, C++ uses the complex operator function and converts the real values to complex values.

## 2.3 Mathematical Functions

The complex arithmetic library provides a number of mathematical functions. Some are peculiar to complex numbers; the rest are complex-number versions of functions in the standard C mathematical library.

All of these functions produce a result for every possible argument. If a function cannot produce a mathematically acceptable result, it calls `complex_error` and returns some suitable value. In particular, the functions try to avoid actual overflow and call `complex_error` with a message instead. The following tables describe the remainder of the complex arithmetic library functions.

### TABLE 2-1 Complex Arithmetic Library Functions

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double abs(const complex)</code></td>
<td>Returns the magnitude of a complex number.</td>
</tr>
<tr>
<td><code>double ang(const complex)</code></td>
<td>Returns the angle of a complex number.</td>
</tr>
<tr>
<td><code>complex conj(const complex)</code></td>
<td>Returns the complex conjugate of its argument.</td>
</tr>
<tr>
<td>Complex Arithmetic Library Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>double imag(const complex&amp;)</td>
<td>Returns the imaginary part of a complex number.</td>
</tr>
<tr>
<td>double norm(const complex)</td>
<td>Returns the square of the magnitude of its argument. Faster than abs, but more likely to cause an overflow. For comparing magnitudes.</td>
</tr>
<tr>
<td>complex polar(double mag, double ang=0.0)</td>
<td>Takes a pair of polar coordinates that represent the magnitude and angle of a complex number and returns the corresponding complex number.</td>
</tr>
<tr>
<td>double real(const complex&amp;)</td>
<td>Returns the real part of a complex number.</td>
</tr>
</tbody>
</table>

### TABLE 2-2  Complex Mathematical and Trigonometric Functions

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex acos(const complex)</td>
<td>Returns the angle whose cosine is its argument.</td>
</tr>
<tr>
<td>complex asin(const complex)</td>
<td>Returns the angle whose sine is its argument.</td>
</tr>
<tr>
<td>complex atan(const complex)</td>
<td>Returns the angle whose tangent is its argument.</td>
</tr>
<tr>
<td>complex cos(const complex)</td>
<td>Returns the cosine of its argument.</td>
</tr>
<tr>
<td>complex csc(const complex)</td>
<td>Returns the hyperbolic cosine of its argument.</td>
</tr>
<tr>
<td>complex exp(const complex)</td>
<td>Computes $e^x$, where $e$ is the base of the natural logarithms, and $x$ is the argument given to exp.</td>
</tr>
<tr>
<td>complex log(const complex)</td>
<td>Returns the natural logarithm of its argument.</td>
</tr>
<tr>
<td>complex log10(const complex)</td>
<td>Returns the common logarithm of its argument.</td>
</tr>
<tr>
<td>complex pow(double b, const complex exp)</td>
<td>Takes two arguments: pow(b, exp). It raises $b$ to the power of $exp$.</td>
</tr>
<tr>
<td>complex pow(const complex b, int exp)</td>
<td></td>
</tr>
<tr>
<td>complex pow(const complex b, double exp)</td>
<td></td>
</tr>
<tr>
<td>complex pow(const complex b, const complex exp)</td>
<td></td>
</tr>
</tbody>
</table>
The complex library has these definitions for error handling:

```cpp
extern int errno;
class c_exception { ... }
int complex_error(c_exception&);
```

The external variable `errno` is the global error state from the C library. `errno` can take on the values listed in the standard header `errno.h` (see the man page `perror(3)`). No function sets `errno` to zero, but many functions set it to other values.

To determine whether a particular operation fails:

1. **Set `errno` to zero before the operation.**
2. **Test the operation.**

   The function `complex_error` takes a reference to type `c_exception` and is called by the following complex arithmetic library functions:
   - `exp`
   - `log`
   - `log10`
   - `sinh`
   - `cosh`
The default version of `complex_error` returns zero. This return of zero means that the default error handling takes place. You can provide your own replacement function `complex_error` that performs other error handling. Error handling is described in the man page `cplxerr(3CC4)`.

Default error handling is described in the man pages `cplxtrig(3CC4)` and `cplxexp(3CC4)` It is also summarized in the following table.

<table>
<thead>
<tr>
<th>Complex Arithmetic Library Function</th>
<th>Default Error Handling Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>exp</code></td>
<td>If overflow occurs, sets <code>errno</code> to <code>ERANGE</code> and returns a huge complex number.</td>
</tr>
<tr>
<td><code>log</code>, <code>log10</code></td>
<td>If the argument is zero, sets <code>errno</code> to <code>EDOM</code> and returns a huge complex number.</td>
</tr>
<tr>
<td><code>sinh</code>, <code>cosh</code></td>
<td>If the imaginary part of the argument causes overflow, returns a complex zero. If the real part causes overflow, returns a huge complex number. In either case, sets <code>errno</code> to <code>ERANGE</code>.</td>
</tr>
</tbody>
</table>

### 2.5 Input and Output

The complex arithmetic library provides default `extractors` and `inserters` for complex numbers, as shown in the following example:

```cpp
ostream& operator<<(ostream&, const complex&); // inserter
istream& operator>>(istream&, complex&); // extractor
```

For basic information on extractors and inserters, see Section 3.2 “Basic Structure of `iostream` Interaction” and Section 3.3.1 “Output Using `iostream`”.

For input, the complex extractor `>>` extracts a pair of numbers (surrounded by parentheses and separated by a comma) from the input stream and reads them into a complex object. The first number is taken as the value of the real part; the second as the value of the imaginary part. For example, given the declaration and input statement:

```cpp
complex x;
cin >> x;
```
and the input \((3.45, 5)\), the value of \(x\) is equivalent to \(3.45 + 5.0i\). The reverse is true for inserters. Given complex \(x(3.45, 5)\), \(\text{cout} \ll x\) prints \((3.45, 5)\).

The input usually consists of a pair of numbers in parentheses separated by a comma; white space is optional. If you provide a single number, with or without parentheses and white space, the extractor sets the imaginary part of the number to zero. Do not include the symbol \(i\) in the input text.

The inserter inserts the values of the real and imaginary parts enclosed in parentheses and separated by a comma. It does not include the symbol \(i\). The two values are treated as doubles.

2.6 Mixed-Mode Arithmetic

Type complex is designed to fit in with the built-in arithmetic types in mixed-mode expressions. Arithmetic types are silently converted to type complex, and there are complex versions of the arithmetic operators and most mathematical functions. For example:

```cpp
int i, j;
double x, y;
complex a, b;
a = sin((b+i)/y) + x/j;
```

The expression \(b+i\) is mixed-mode. Integer \(i\) is converted to type complex via the constructor complex::complex(double, double=0), the integer first being converted to type double. The result is to be divided by \(y\), a double, so \(y\) is also converted to complex and the complex divide operation is used. The quotient is thus type complex, so the complex sine routine is called, yielding another complex result, and so on.

Not all arithmetic operations and conversions are implicit, or even defined, however. For example, complex numbers are not well-ordered, mathematically speaking, and complex numbers can be compared for equality only.

```cpp
complex a, b;
a == b; // OK
a != b; // OK
a < b; // error: operator < cannot be applied to type complex
a >= b; // error: operator >= cannot be applied to type complex
```
Similarly, there is no automatic conversion from type complex to any other type, because the concept is not well-defined. You can specify whether you want the real part, imaginary part, or magnitude, for example.

```c
complex a;
double f(double);
f(abs(a)); // OK
f(a);     // error: no match for f(complex)
```

## 2.7 Efficiency

The design of the complex class addresses efficiency concerns.

The simplest functions are declared inline to eliminate function call overhead.

Several overloaded versions of functions are provided when that makes a difference. For example, the `pow` function has versions that take exponents of type `double` and `int` as well as `complex`, since the computations for the former are much simpler.

The standard C math library header `math.h` is included automatically when you include `complex.h`. The C++ overloading rules then result in efficient evaluation of expressions like this:

```c
double x;
complex x = sqrt(x);
```

In this example, the standard math function `sqrt(double)` is called, and the result is converted to type `complex`, rather than converting to type `complex` first and then calling `sqrt(complex)`. This result falls right out of the overload resolution rules, and is precisely the result you want.
2.8 Complex Man Pages

The remaining documentation of the complex arithmetic library consists of the man pages listed in the following table.

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>cplx.intro(3CC4)</td>
<td>General introduction to the complex arithmetic library</td>
</tr>
<tr>
<td>cartpol(3CC4)</td>
<td>Cartesian and polar functions</td>
</tr>
<tr>
<td>cplxerr(3CC4)</td>
<td>Error-handling functions</td>
</tr>
<tr>
<td>cplxexp(3CC4)</td>
<td>Exponential, log, and square root functions</td>
</tr>
<tr>
<td>cplxops(3CC4)</td>
<td>Arithmetic operator functions</td>
</tr>
<tr>
<td>cplxtrig(3CC4)</td>
<td>Trigonometric functions</td>
</tr>
</tbody>
</table>
The Classic *iostream* Library

C++, like C, has no built-in input or output statements. Instead, I/O facilities are provided by a library. The Sun WorkShop 6 C++ compiler provides both the classic implementation and the ISO standard implementation of the *iostream* classes.

- In compatibility mode (`-compat[4]`), the classic *iostream* classes are contained in `libC`.
- In standard mode (default mode), the classic *iostream* classes are contained in `libiostream`. Use `libiostream` when you have source code that uses the classic *iostream* classes and you want to compile the source in standard mode. To use the classic *iostream* facilities in standard mode, include the `iostream.h` header file and compile using the `-library=iostream` option.
- The standard *iostream* classes are available only in standard mode, and are contained in the C++ standard library, `libCstd`.

This chapter provides an introduction to the classic *iostream* library and provides examples of its use. This chapter does not provide a complete description of the *iostream* library. See the *iostream* library man pages for more details. To access the classic *iostream* man pages type:

```
example% man -s 3CC4 name
```

### 3.1 Predefined *iostreams*

There are four predefined *iostreams*:
- `cin`, connected to standard input
- `cout`, connected to standard output
- `cerr`, connected to standard error
- `clog`, connected to standard error
The predefined \texttt{iostreams} are fully buffered, except for \texttt{cerr}. See Section 3.3.1 “Output Using \texttt{iostream}” and Section 3.3.2 “Input Using \texttt{iostream}”.

3.2 Basic Structure of \texttt{iostream} Interaction

By including the \texttt{iostream} library, a program can use any number of input or output streams. Each stream has some source or sink, which may be one of the following:

- Standard input
- Standard output
- Standard error
- A file
- An array of characters

A stream can be restricted to input or output, or a single stream can allow both input and output. The \texttt{iostream} library implements these streams using two processing layers.

- The lower layer implements sequences, which are simply streams of characters. These sequences are implemented by the \texttt{streambuf} class, or by classes derived from it.
- The upper layer performs formatting operations on sequences. These formatting operations are implemented by the \texttt{istream} and \texttt{ostream} classes, which have as a member an object of a type derived from class \texttt{streambuf}. An additional class, \texttt{iostream}, is for streams on which both input and output can be performed.

Standard input, output, and error are handled by special class objects derived from class \texttt{istream} or \texttt{ostream}.

The \texttt{ifstream}, \texttt{ofstream}, and \texttt{fstream} classes, which are derived from \texttt{istream}, \texttt{ostream}, and \texttt{iostream} respectively, handle input and output with files.

The \texttt{istrstream}, \texttt{ostrstream}, and \texttt{strstream} classes, which are derived from \texttt{istream}, \texttt{ostream}, and \texttt{iostream} respectively, handle input and output to and from arrays of characters.

When you open an input or output stream, you create an object of one of these types, and associate the \texttt{streambuf} member of the stream with a device or file. You generally do this association through the stream constructor, so you don’t work with the \texttt{streambuf} directly. The \texttt{iostream} library predefines stream objects for the standard input, standard output, and error output, so you don’t have to create your own objects for those streams.
You use operators or istream member functions to insert data into a stream (output) or extract data from a stream (input), and to control the format of data that you insert or extract.

When you want to insert and extract a new data type—one of your classes—you generally overload the insertion and extraction operators.

### 3.3 Using the Classic istream Library

To use routines from the classic istream library, you must include the header files for the part of the library you need. The header files are described in the following table.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>istream.h</td>
<td>Declares basic features of istream library.</td>
</tr>
<tr>
<td>fstream.h</td>
<td>Declares istreams and streambufs specialized to files. Includes istream.h.</td>
</tr>
<tr>
<td>strstream.h</td>
<td>Declares istreams and streambufs specialized to character arrays. Includes istream.h.</td>
</tr>
<tr>
<td>iomanip.h</td>
<td>Declares manipulators: values you insert into or extract from istreams to have different effects. Includes istream.h.</td>
</tr>
<tr>
<td>stdiostream.h</td>
<td>(obsolete) Declares istreams and streambufs specialized to use stdio FILES. Includes istream.h.</td>
</tr>
<tr>
<td>stream.h</td>
<td>(obsolete) Includes istream.h, fstream.h, iomanip.h, and stdiostream.h. For compatibility with old-style streams from C++ version 1.2.</td>
</tr>
</tbody>
</table>

You usually do not need all of these header files in your program. Include only the ones that contain the declarations you need. In compatibility mode (-compat[=4]), the classic istream library is part of libC, and is linked automatically by the CC driver. In standard mode (the default), libiostream contains the classic istream library.
3.3.1 Output Using iostream

Output using iostream usually relies on the overloaded left-shift operator (<<) which, in the context of iostream, is called the insertion operator. To output a value to standard output, you insert the value in the predefined output stream cout. For example, given a value someValue, you send it to standard output with a statement like:

```
cout << someValue;
```

The insertion operator is overloaded for all built-in types, and the value represented by someValue is converted to its proper output representation. If, for example, someValue is a float value, the << operator converts the value to the proper sequence of digits with a decimal point. Where it inserts float values on the output stream, << is called the float inserter. In general, given a type X, << is called the X inserter. The format of output and how you can control it is discussed in the ios(3CC4) man page.

The iostream library does not support user-defined types. If you define types that you want to output in your own way, you must define an inserter (that is, overload the << operator) to handle them correctly.

The << operator can be applied repetitively. To insert two values on cout, you can use a statement like the one in the following example:

```
cout << someValue << anotherValue;
```

The output from the above example will show no space between the two values. So you may want to write the code this way:

```
cout << someValue << " " << anotherValue;
```

The << operator has the precedence of the left shift operator (its built-in meaning). As with other operators, you can always use parentheses to specify the order of action. It is often a good idea to use parentheses to avoid problems of precedence. Of the following four statements, the first two are equivalent, but the last two are not.

```
cout << a+b;  // + has higher precedence than <<
cout << (a+b);
cout << (a&y); // << has precedence higher than &
cout << a&y;   // probably an error: (cout << a) & y
```
3.3.1.1 Defining Your Own Insertion Operator

The following example defines a string class:

```cpp
#include <stdlib.h>
#include <iostream.h>

class string {
private:
    char* data;
    size_t size;

public:
    // (functions not relevant here)

    friend ostream& operator<<(ostream&, const string&);
    friend istream& operator>>(istream&, string&);
};

ostream& operator<<(ostream& ostr, const string& output)
{
    return ostr << output.data;
}
```

The insertion and extraction operators must in this case be defined as friends because the data part of the string class is private.

```cpp
cout << string1 << string2;
```

Here is the definition of operator<< overloaded for use with strings.

operator<< takes ostream& (that is, a reference to an ostream) as its first argument and returns the same ostream, making it possible to combine insertions in one statement.

3.3.1.2 Handling Output Errors

Generally, you don’t have to check for errors when you overload operator<< because the iostream library is arranged to propagate errors.

When an error occurs, the iostream where it occurred enters an error state. Bits in the iostream's state are set according to the general category of the error. The inserters defined in iostream ignore attempts to insert data into any stream that is in an error state, so such attempts do not change the iostream's state.
In general, the recommended way to handle errors is to periodically check the state of the output stream in some central place. If there is an error, you should handle it in some way. This chapter assumes that you define a function `error`, which takes a string and aborts the program. `error` is not a predefined function. See Section 3.3.9 “Handling Input Errors” for an example of an `error` function. You can examine the state of an `iostream` with the operator `!`, which returns a nonzero value if the `iostream` is in an error state. For example:

```cpp
if (!cout) error( "output error");
```

There is another way to test for errors. The `ios` class defines `operator void *()`, so it returns a NULL pointer when there is an error. You can use a statement like:

```cpp
if (cout << x) return ; // return if successful
```

You can also use the function `good`, a member of `ios`:

```cpp
if ( cout.good() ) return ; // return if successful
```

The error bits are declared in the `enum`:

```cpp
enum io_state { goodbit=0, eofbit=1, failbit=2, badbit=4, hardfail=0x80} ;
```

For details on the error functions, see the `iostream` man pages.

### 3.3.1.3 Flushing

As with most I/O libraries, `iostream` often accumulates output and sends it on in larger and generally more efficient chunks. If you want to flush the buffer, you simply insert the special value `flush`. For example:

```cpp
cout << "This needs to get out immediately." << flush ;
```

`flush` is an example of a kind of object known as a `manipulator`, which is a value that can be inserted into an `iostream` to have some effect other than causing output of its value. It is really a function that takes an `ostream&` or `istream&` argument and returns its argument after performing some actions on it (see Section 3.7 “Manipulators”).
### 3.3.1.4 Binary Output

To obtain output in the raw binary form of a value, use the member function `write` as shown in the following example. This example shows the output in the raw binary form of `x`.

```cpp
cout.write((char*)&x, sizeof(x));
```

The previous example violates type discipline by converting `&x` to `char*`. Doing so is normally harmless, but if the type of `x` is a class with pointers, virtual member functions, or one that requires nontrivial constructor actions, the value written by the above example cannot be read back in properly.

### 3.3.2 Input Using `iostream`

Input using `iostream` is similar to output. You use the extraction operator `>>` and you can string together extractions the way you can with insertions. For example:

```cpp
cin >> a >> b;
```

This statement gets two values from standard input. As with other overloaded operators, the extractors used depend on the types of `a` and `b` (and two different extractors are used if `a` and `b` have different types). The format of input and how you can control it is discussed in some detail in the `ios(3CC4)` man page. In general, leading whitespace characters (spaces, newlines, tabs, form-feeds, and so on) are ignored.

### 3.3.3 Defining Your Own Extraction Operators

When you want input for a new type, you overload the extraction operator for it, just as you overload the insertion operator for output.

Class `string` defines its extraction operator in the following code example:

```cpp
CODE EXAMPLE 3-1 string Extraction Operator

```cpp
istream& operator>> (istream& istr, string& input)
{
    const int maxline = 256;
    char holder[maxline];
    // More code...
}
```
The `get` function reads characters from the input stream `istr` and stores them in `holder` until `maxline-1` characters have been read, or a new line is encountered, or EOF, whichever happens first. The data in `holder` is then null-terminated. Finally, the characters in `holder` are copied into the target string.

By convention, an extractor converts characters from its first argument (in this case, `istream& istr`), stores them in its second argument, which is always a reference, and returns its first argument. The second argument must be a reference because an extractor is meant to store the input value in its second argument.

### 3.3.4 Using the `char*` Extractor

This predefined extractor is mentioned here because it can cause problems. Use it like this:

```cpp
char x[50];
cin >> x;
```

This extractor skips leading whitespace and extracts characters and copies them to `x` until it reaches another whitespace character. It then completes the string with a terminating null (0) character. Be careful, because input can overflow the given array.

You must also be sure the pointer points to allocated storage. For example, here is a common error:

```cpp
char * p; // not initialized
cin >> p;
```

There is no telling where the input data will be stored, and it may cause your program to abort.
3.3.5 Reading Any Single Character

In addition to using the char extractor, you can get a single character with either form of the get member function. For example:

```cpp
char c;
cin.get(c); // leaves c unchanged if input fails

int b;
b = cin.get(); // sets b to EOF if input fails
```

**Note** – Unlike the other extractors, the char extractor does not skip leading whitespace.

Here is a way to skip only blanks, stopping on a tab, newline, or any other character:

```cpp
int a;
do {
    a = cin.get();
} while (a == ' ');
```

3.3.6 Binary Input

If you need to read binary values (such as those written with the member function write), you can use the read member function. The following example shows how to input the raw binary form of x using the read member function, and is the inverse of the earlier example that uses write.

```cpp
cin.read((char*)x, sizeof(x));
```

3.3.7 Peeking at Input

You can use the peek member function to look at the next character in the stream without extracting it. For example:

```cpp
if (cin.peek() != c) return 0;
```
3.3.8 Extracting Whitespace

By default, the `iostream` extractors skip leading whitespace. You can turn off the `skip` flag to prevent this from happening. The following example turns off whitespace skipping from `cin`, then turns it back on:

```cpp
cin.unsetf(ios::skipws); // turn off whitespace skipping
...
cin.setf(ios::skipws); // turn it on again
```

You can use the `iostream` manipulator `ws` to remove leading whitespace from the `iostream`, whether or not skipping is enabled. The following example shows how to remove the leading whitespace from `iostream istr`:

```cpp
istr >> ws;
```

3.3.9 Handling Input Errors

By convention, an extractor whose first argument has a nonzero error state should not extract anything from the input stream and should not clear any error bits. An extractor that fails should set at least one error bit.

As with output errors, you should check the error state periodically and take some action, such as aborting, when you find a nonzero state. The `!` operator tests the error state of an `iostream`. For example, the following code produces an input error if you type alphabetic characters for input:

```cpp
#include <unistd.h>
#include <iostream.h>

void error (const char* message) {
    cerr << message << "\n";
    exit(1);
}

int main() {
    cout << "Enter some characters: ";
    int bad;
    cin >> bad;
    if (!cin) error("aborted due to input error");
    cout << "If you see this, not an error." << "\n";
    return 0;
}
```
Class `ios` has member functions that you can use for error handling. See the man pages for details.

### 3.3.10 Using `iostreams` with `stdio`

You can use `stdio` with C++ programs, but problems can occur when you mix `iostreams` and `stdio` in the same standard stream within a program. For example, if you write to both `stdout` and `cout`, independent buffering occurs and produces unexpected results. The problem is worse if you input from both `stdin` and `cin`, since independent buffering may turn the input into trash.

To eliminate this problem with standard input, standard output and standard error, use the following instruction before performing any input or output. It connects all the predefined `iostreams` with the corresponding predefined `stdio` FILES.

```cpp
ios::sync_with_stdio();
```

Such a connection is not the default because there is a significant performance penalty when the predefined streams are made unbuffered as part of the connection. You can use both `stdio` and `iostreams` in the same program applied to different files. That is, you can write to `stdout` using `stdio` routines and write to other files attached to `iostreams`. You can open `stdio` FILES for input and also read from `cin` so long as you don’t also try to read from `stdin`.

### 3.4 Creating `iostreams`

To read or write a stream other than the predefined `iostreams`, you need to create your own `iostream`. In general, that means creating objects of types defined in the `iostream` library. This section discusses the various types available.

#### 3.4.1 Dealing with Files Using Class `fstream`

Dealing with files is similar to dealing with standard input and standard output; classes `ifstream`, `ofstream`, and `fstream` are derived from classes `istream`, `ostream`, and `iostream`, respectively. As derived classes, they inherit the insertion and extraction operations (along with the other member functions) and also have members and constructors for use with files.
Include the file `fstream.h` to use any of the `fstreams`. Use an `ifstream` when you only want to perform input, an `ofstream` for output only, and an `fstream` for a stream on which you want to perform both input and output. Use the name of the file as the constructor argument.

For example, copy the file `thisFile` to the file `thatFile` as in the following example:

```cpp
#include <fstream>

int main()
{
    ifstream fromFile("thisFile");
    if (!fromFile)
        error("unable to open ‘thisFile’ for input");
    ofstream toFile("thatFile");
    if (!toFile)
        error("unable to open ‘thatFile’ for output");
    char c;
    while (toFile && fromFile.get(c)) toFile.put(c);
    return 0;
}
```

This code:
- Creates an `ifstream` object called `fromFile` with a default mode of `ios::in` and connects it to `thisFile`. It opens `thisFile`.
- Checks the error state of the new `ifstream` object and, if it is in a failed state, calls the `error` function, which must be defined elsewhere in the program.
- Creates an `ofstream` object called `toFile` with a default mode of `ios::out` and connects it to `thatFile`.
- Checks the error state of `toFile` as above.
- Creates a `char` variable to hold the data while it is passed.
- Copies the contents of `fromFile` to `toFile` one character at a time.

**Note** — It is, of course, undesirable to copy a file this way, one character at a time. This code is provided just as an example of using `fstreams`. You should instead insert the streambuf associated with the input stream into the output stream. See Section 3.10 "Streambufs", and the man page `sbufpub(3CC4)`.

### 3.4.1.1 Open Mode

The mode is constructed by or-ing bits from the enumerated type `open_mode`, which is a public type of class `ios` and has the definition:

```cpp
enum open_mode {binary=0, in=1, out=2, ate=4, app=8, trunc=0x10,
    nocreate=0x20, noreplace=0x40};
```
Note – The binary flag is not needed on Unix, but is provided for compatibility with systems that do need it. Portable code should use the binary flag when opening binary files.

You can open a file for both input and output. For example, the following code opens file someName for both input and output, attaching it to the fstream variable inoutFile.

```cpp
fstream inoutFile("someName", ios::in|ios::out);
```

### 3.4.1.2 Declaring an fstream Without Specifying a File

You can declare an fstream without specifying a file and open the file later. For example, the following creates the ofstream toFile for writing.

```cpp
ofstream toFile;
toFile.open(argv[1], ios::out);
```

### 3.4.1.3 Opening and Closing Files

You can close the fstream and then open it with another file. For example, to process a list of files provided on the command line:

```cpp
ifstream infile;
for (char** f = &argv[1]; *f; ++f) {
    infile.open(*f, ios::in);
    ...
    infile.close();
}
```

### 3.4.1.4 Opening a File Using a File Descriptor

If you know a file descriptor, such as the integer 1 for standard output, you can open it like this:

```cpp
ofstream outfile;
outfile.attach(1);
```
When you open a file by providing its name to one of the `fstream` constructors or by using the `open` function, the file is automatically closed when the `fstream` is destroyed (by a `delete` or when it goes out of scope). When you attach a file to an `fstream`, it is not automatically closed.

### 3.4.1.5 Repositioning within a File

You can alter the reading and writing position in a file. Several tools are supplied for this purpose.

- `streampos` is a type that can record a position in an `iostream`.
- `tellg (tellp)` is an `istream (ostream)` member function that reports the file position. Since `istream` and `ostream` are the parent classes of `fstream`, `tellg` and `tellp` can also be invoked as a member function of the `fstream` class.
- `seekg (seekp)` is an `istream (ostream)` member function that finds a given position.
- The `seek_dir` enum specifies relative positions for use with `seek`.

```cpp
enum seek_dir { beg=0, cur=1, end=2 }
```

For example, given an `fstream` `aFile`:

```cpp
streampos original = aFile.tellp(); //save current position
aFile.seekp(0, ios::end); //reposition to end of file
aFile << x;               //write a value to file
aFile.seekp(original);    //return to original position
```

`seekg (seekp)` can take one or two parameters. When it has two parameters, the first is a position relative to the position indicated by the `seek_dir` value given as the second parameter. For example:

```cpp
aFile.seekp(-10, ios::end);
```

moves to 10 bytes from the end while

```cpp
aFile.seekp(10, ios::cur);
```

moves to 10 bytes forward from the current position.
Note – Arbitrary seeks on text streams are not portable, but you can always return to a previously saved streampos value.

3.5 Assignment of iostreams

iostreams does not allow assignment of one stream to another.

The problem with copying a stream object is that there are then two versions of the state information, such as a pointer to the current write position within an output file, which can be changed independently. As a result, problems could occur.

3.6 Format Control

Format control is discussed in detail in the man page ios(3CC4).

3.7 Manipulators

Manipulators are values that you can insert into or extract from iostreams to have special effects.

Parameterized manipulators are manipulators that take one or more parameters.

Because manipulators are ordinary identifiers, and therefore use up possible names, iostream doesn’t define them for every possible function. A number of manipulators are discussed with member functions in other parts of this chapter.

There are 13 predefined manipulators, as described in TABLE 3-2. When using that table, assume the following:

- i has type long.
- n has type int.
- c has type char.
- istr is an input stream.
- `ostr` is an output stream.

### TABLE 3-2  iostream Predefined Manipulators

<table>
<thead>
<tr>
<th>Predefined Manipulator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ostr &lt;&lt; dec, istr &gt;&gt; dec</td>
<td>Makes the integer conversion base 10.</td>
</tr>
</tbody>
</table>
| 2 ostr << endl | Inserts a newline character (`'\n'`) and invokes `ostream::flush()`.
| 3 ostr << ends | Inserts a null (0) character. Useful when dealing with `strstreams`.
| 4 ostr << flush | Invokes `ostream::flush()`.
| 5 ostr << hex, istr >> hex | Makes the integer conversion base 16. |
| 6 ostr << oct, istr >> oct | Make the integer conversion base 8. |
| 7 istr >> ws | Extracts whitespace characters (skips whitespace) until a non-whitespace character is found (which is left in `istr`). |
| 8 ostr << setbase(n), istr >> setbase(n) | Sets the conversion base to `n` (0, 8, 10, 16 only). |
| 9 ostr << setw(n), istr >> setw(n) | Invokes `ios::width(n)`. Sets the field width to `n`. |
| 10 ostr << resetiosflags(i), istr >> resetiosflags(i) | Clears the flags bitvector according to the bits set in `i`. |
| 11 ostr << setiosflags(i), istr >> setiosflags(i) | Sets the flags bitvector according to the bits set in `i`. |
| 12 ostr << setfill(c), istr >> setfill(c) | Sets the fill character (for padding a field) to `c`. |
| 13 ostr << setprecision(n), istr >> setprecision(n) | Sets the floating-point precision to `n` digits. |

To use predefined manipulators, you must include the file `iomanip.h` in your program.

You can define your own manipulators. There are two basic types of manipulator:

- Plain manipulator—Takes an `istream&`, `ostream&`, or `ios&` argument, operates on the stream, and then returns its argument.
- Parameterized manipulator—Takes an `istream&`, `ostream&`, or `ios&` argument, one additional argument (the parameter), operates on the stream, and then returns its stream argument.
3.7.1 Using Plain Manipulators

A plain manipulator is a function that:
- Takes a reference to a stream
- Operates on it in some way
- Returns its argument

The shift operators taking (a pointer to) such a function are predefined for iostreams, so the function can be put in a sequence of input or output operators. The shift operator calls the function rather than trying to read or write a value. An example of a tab manipulator that inserts a tab in an ostream is:

```cpp
ostream& tab(ostream& os) {
    return os << '	';
}
...
cout << x << tab << y;
```

This is an elaborate way to achieve the following:

```cpp
const char tab = '\t';
...
cout << x << tab << y;
```
Here is another example, which cannot be accomplished with a simple constant. Suppose you want to turn whitespace skipping on and off for an input stream. You can use separate calls to `ios::setf` and `ios::unsetf` to turn the `skipws` flag on and off, or you could define two manipulators:

```cpp
#include <iostream.h>
#include <iomanip.h>

istream& skipon(istream &is) {
    is.setf(ios::skipws, ios::skipws);
    return is;
}

istream& skipoff(istream& is) {
    is.unsetf(ios::skipws);
    return is;
}

int main ()
{
    int x,y;
    cin >> skipon >> x >> skipoff >> y;
    return 1;
}
```

### 3.7.2 Parameterized Manipulators

One of the parameterized manipulators that is included in `<iomanip.h>` is `setfill`. `setfill` sets the character that is used to fill out field widths. It is implemented as shown in the following example:

```cpp
#include<iostream.h>
#include<iomanip.h>

static ios & sfill(ios& i, int f) {
    i.fill(f);
    return i;
}

smanip_int setfill(int f) {
    return smanip_int(sfill, f);
}
```

A parameterized manipulator is implemented in two parts:
The manipulator. It takes an extra parameter. In the previous code example, it takes an extra int parameter. You cannot place this manipulator function in a sequence of input or output operations, since there is no shift operator defined for it. Instead, you must use an auxiliary function, the applicator.

The applicator. It calls the manipulator. The applicator is a global function, and you make a prototype for it available in a header file. Usually the manipulator is a static function in the file containing the source code for the applicator. The manipulator is called only by the applicator, and if you make it static, you keep its name out of the global address space.

Several classes are defined in the header file iomanip.h. Each class holds the address of a manipulator function and the value of one parameter. The iomanip classes are described in the man page manip(3CC4). The previous example uses the smanip_int class, which works with an ios. Because it works with an ios, it also works with an istream and an ostream. The previous example also uses a second parameter of type int.

The applicator creates and returns a class object. In the previous code example the class object is an smanip_int, and it contains the manipulator and the int argument to the applicator. The iomanip.h header file defines the shift operators for this class. When the applicator function setfill appears in a sequence of input or output operations, the applicator function is called, and it returns a class. The shift operator acts on the class to call the manipulator function with its parameter value, which is stored in the class.

In the following example, the manipulator print_hex:

- Puts the output stream into the hex mode.
- Inserts a long value into the stream.
- Restores the conversion mode of the stream.

The class omanip_long is used because this code example is for output only, and it operates on a long rather than an int:

```cpp
#include <iostream.h>
#include <iomanip.h>

static ostream& xfield(ostream& os, long v) {
    long save = os.setf(ios::hex, ios::basefield);
    os << v;
    os.setf(save, ios::basefield);
    return os;
}

omanip_long print_hex(long v) {
    return omanip_long(xfield, v);
}
```
3.8 **Strstreams: iostreams for Arrays**

See the `strstream(3CC4)` man page.

3.9 **Stdiobufs: iostreams for stdio Files**

See the `stdiobuf(3CC4)` man page.

3.10 **Streambufs**

`iostreams` are the formatting part of a two-part (input or output) system. The other part of the system is made up of `streambufs`, which deal in input or output of unformatted streams of characters.

You usually use `streambufs` through `iostreams`, so you don’t have to worry about the details of `streambufs`. You can use `streambufs` directly if you choose to, for example, if you need to improve efficiency or to get around the error handling or formatting built into `iostreams`.

3.10.1 **Working with Streambufs**

A `streambuf` consists of a stream or sequence of characters and one or two pointers into that sequence. Each pointer points between two characters. (Pointers cannot actually point between characters, but it is helpful to think of them that way.) There are two kinds of `streambuf` pointers:

- A `put` pointer, which points just before the position where the next character will be stored
- A `get` pointer, which points just before the next character to be fetched

A `streambuf` can have one or both of these pointers.
3.10.1.1 Position of Pointers

The positions of the pointers and the contents of the sequences can be manipulated in various ways. Whether or not both pointers move when manipulated depends on the kind of streambuf used. Generally, with queue-like streambufs, the get and put pointers move independently; with file-like streambufs the get and put pointers always move together. A strstream is an example of a queue-like stream; an fstream is an example of a file-like stream.

3.10.2 Using Streambufs

You never create an actual streambuf object, but only objects of classes derived from class streambuf. Examples are filebuf and strstreambuf, which are described in man pages filebuf(3CC4) and ssbuf(3), respectively. Advanced users may want to derive their own classes from streambuf to provide an interface to a special device or to provide other than basic buffering. Man pages sbufpub(3CC4) and sbufprot(3CC4) discuss how to do this.

Apart from creating your own special kind of streambuf, you may want to access the streambuf associated with an iostream to access the public member functions, as described in the man pages referenced above. In addition, each iostream has a defined inserter and extractor which takes a streambuf pointer. When a streambuf is inserted or extracted, the entire stream is copied.

Here is another way to do the file copy discussed earlier, with the error checking omitted for clarity:

```cpp
#include <iostream>

int main() {
    std::ifstream fromFile("thisFile");
    std::ofstream toFile("thatFile");
    toFile << fromFile.rdbuf();
    fromFile >> toFile.rdbuf();
    return 0;
}
```

We open the input and output files as before. Every iostream class has a member function rdbuf that returns a pointer to the streambuf object associated with it. In the case of an fstream, the streambuf object is type filebuf. The entire file associated with fromFile is copied (inserted into) the file associated with toFile. The last line could also be written like this:

```cpp
fromFile >> toFile.rdbuf();
```

The source file is then extracted into the destination. The two methods are entirely equivalent.
3.11 iostream Man Pages

A number of C++ man pages give details of the iostream library. The following table gives an overview of what is in each man page.

To access a classic iostream library man page, type:

```
example% man -s 3CC4 name
```

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>filebuf</td>
<td>Details the public interface for the class filebuf, which is derived from streambuf and is specialized for use with files. See the sbufpub(3CC4) and sbufprot(3CC4) man pages for details of features inherited from class streambuf. Use the filebuf class through class fstream.</td>
</tr>
<tr>
<td>fstream</td>
<td>Details specialized member functions of classes ifstream, ofstream, and fstream, which are specialized versions of istream, ostream, and iostream for use with files.</td>
</tr>
<tr>
<td>ios</td>
<td>Details parts of class ios, which functions as a base class for iostreams. It contains state data common to all streams.</td>
</tr>
<tr>
<td>ios.intro</td>
<td>Gives an introduction to and overview of iostreams.</td>
</tr>
<tr>
<td>istream</td>
<td>Details the following:</td>
</tr>
<tr>
<td></td>
<td>• Member functions for class istream, which supports interpretation of characters fetched from a streambuf</td>
</tr>
<tr>
<td></td>
<td>• Input formatting</td>
</tr>
<tr>
<td></td>
<td>• Positioning functions described as part of class ostream.</td>
</tr>
<tr>
<td></td>
<td>• Some related functions</td>
</tr>
<tr>
<td></td>
<td>• Related manipulators</td>
</tr>
<tr>
<td>manip</td>
<td>Describes the input and output manipulators defined in the iostream library.</td>
</tr>
<tr>
<td>ostream</td>
<td>Details the following:</td>
</tr>
<tr>
<td></td>
<td>• Member functions for class ostream, which supports interpretation of characters written to a streambuf</td>
</tr>
<tr>
<td></td>
<td>• Output formatting</td>
</tr>
<tr>
<td></td>
<td>• Positioning functions described as part of class ostream</td>
</tr>
<tr>
<td></td>
<td>• Some related functions</td>
</tr>
<tr>
<td></td>
<td>• Related manipulators</td>
</tr>
</tbody>
</table>
TABLE 3-3  iostream Man Pages Overview (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbufprot</td>
<td>Describes the interface needed by programmers who are coding a class derived from class streambuf. Also refer to the sbufpub(3C4) man page because some public functions are not discussed in the sbufprot(3C4) man page.</td>
</tr>
<tr>
<td>sbufpub</td>
<td>Details the public interface of class streambuf, in particular, the public member functions of streambuf. This man page contains the information needed to manipulate a streambuf-type object directly, or to find out about functions that classes derived from streambuf inherit from it. If you want to derive a class from streambuf, also see the sbufprot(3C4) man page.</td>
</tr>
<tr>
<td>ssbuf</td>
<td>Details the specialized public interface of class strstreambuf, which is derived from streambuf and specialized for dealing with arrays of characters. See the sbufpub(3C4) man page for details of features inherited from class streambuf.</td>
</tr>
<tr>
<td>stdiobuf</td>
<td>Contains a minimal description of class stdiobuf, which is derived from stdiobuf and specialized for dealing with stdio FILEs. See the sbufpub(3C4) man page for details of features inherited from class streambuf.</td>
</tr>
<tr>
<td>strstream</td>
<td>Details the specialized member functions of strstreams, which are implemented by a set of classes derived from the iostream classes and specialized for dealing with arrays of characters.</td>
</tr>
</tbody>
</table>
### 3.12 iostream Terminology

The iostream library descriptions often use terms similar to terms from general programming, but with specialized meanings. The following table defines these terms as they are used in discussing the iostream library.

<table>
<thead>
<tr>
<th><strong>iostream Term</strong></th>
<th><strong>Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer</td>
<td>A word with two meanings, one specific to the iostream package and one more generally applied to input and output. When referring specifically to the iostream library, a buffer is an object of the type defined by the class <code>streambuf</code>. A buffer, generally, is a block of memory used to make efficient transfer of characters for input of output. With buffered I/O, the actual transfer of characters is delayed until the buffer is full or forcibly flushed. An unbuffered buffer refers to a <code>streambuf</code> where there is no buffer in the general sense defined above. This chapter avoids use of the term buffer to refer to <code>streambufs</code>. However, the man pages and other C++ documentation do use the term buffer to mean <code>streambufs</code>.</td>
</tr>
<tr>
<td>Extraction</td>
<td>The process of taking input from an iostream.</td>
</tr>
<tr>
<td>Fstream</td>
<td>An input or output stream specialized for use with files. Refers specifically to a class derived from class <code>iostream</code> when printed in courier font.</td>
</tr>
<tr>
<td>Insertion</td>
<td>The process of sending output into an iostream.</td>
</tr>
<tr>
<td>iostream</td>
<td>Generally, an input or output stream.</td>
</tr>
<tr>
<td>iostream library</td>
<td>The library implemented by the include files <code>iostream.h</code>, <code>fstream.h</code>, <code>strstream.h</code>, <code>iomanip.h</code>, and <code>stdio.h</code>. Because iostream is an object-oriented library, you should extend it. So, some of what you can do with the iostream library is not implemented.</td>
</tr>
<tr>
<td>Stream</td>
<td>An <code>iostream</code>, <code>fstream</code>, <code>strstream</code>, or user-defined stream in general.</td>
</tr>
<tr>
<td>Streambuf</td>
<td>A buffer that contains a sequence of characters with a put or get pointer, or both. When printed in courier font, it means the particular class. Otherwise, it refers generally to any object of class <code>streambuf</code> or a class derived from <code>streambuf</code>. Any stream object contains an object, or a pointer to an object, of a type derived from <code>streambuf</code>.</td>
</tr>
<tr>
<td>Strstream</td>
<td>An <code>iostream</code> specialized for use with character arrays. It refers to the specific class when printed in courier font.</td>
</tr>
</tbody>
</table>
Using Classic iostreams in a Multithreading Environment

This chapter describes how to use the iostream classes of the libC and libiostream libraries for input-output (I/O) in a multithreaded environment. It also provides examples of how to extend functionality of the library by deriving from the iostream classes. This chapter is not a guide for writing multithreaded code in C++, however.

The discussion here applies only to the old iostreams (libC and libiostream) and does not apply to libCstd, the new iostream that is part of the C++ Standard Library.

4.1 Multithreading

Multithreading (MT) is a powerful facility that can speed up applications on multiprocessors; it can also simplify the structuring of applications on both multiprocessors and uniprocessors. The iostream library has been modified to allow its interfaces to be used by applications in a multithreaded environment by programs that utilize the multithreading capabilities when running Solaris version 2.6, 7, or 8 of the Solaris operating environment. Applications that utilize the single-threaded capabilities of previous versions of the library are not affected by the behavior of the modified iostream interfaces.

A library is defined to be MT-safe if it works correctly in an environment with threads. Generally, this “correctness” means that all of its public functions are reentrant. The iostream library provides protection against multiple threads that attempt to modify the state of objects (that is, instances of a C++ class) shared by more than one thread. However, the scope of MT-safety for an iostream object is confined to the period in which the object’s public member function is executing.
Caution – An application is not automatically guaranteed to be MT-safe because it uses MT-safe objects from the libc library. An application is defined to be MT-safe only when it executes as expected in a multithreaded environment.

4.2 Organization of the MT-Safe iostream Library

The organization of the MT-safe iostream library is slightly different from other versions of the iostream library. The exported interface of the library refers to the public and protected member functions of the iostream classes and the set of base classes available, and is consistent with other versions; however, the class hierarchy is different. See Section 4.3 “Interface Changes to the iostream Library” for details.

The original core classes have been renamed with the prefix unsafe_. TABLE 4-1 lists the classes that are the core of the iostream package.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream_MT</td>
<td>The base class for MT-safe classes.</td>
</tr>
<tr>
<td>streambuf</td>
<td>The base class for buffers.</td>
</tr>
<tr>
<td>unsafe_ios</td>
<td>A class that contains state variables that are common to the various stream classes; for example, error and formatting state.</td>
</tr>
<tr>
<td>unsafe_istream</td>
<td>A class that supports formatted and unformatted conversion from sequences of characters retrieved from the streambufs.</td>
</tr>
<tr>
<td>unsafe_ostream</td>
<td>A class that supports formatted and unformatted conversion to sequences of characters stored into the streambufs.</td>
</tr>
<tr>
<td>unsafe_iostream</td>
<td>A class that combines unsafe_istream and unsafe_ostream classes for bidirectional operations.</td>
</tr>
</tbody>
</table>

Each MT-safe class is derived from the base class stream_MT. Each MT-safe class, except streambuf, is also derived from the existing unsafe_base class. Here are some examples:

class streambuf: public stream_MT { ...;}
class ios: virtual public unsafe_ios, public stream_MT { ...;}
class istream: virtual public ios, public unsafe_istream { ...;}
The class `stream_MT` provides the mutual exclusion (mutex) locks required to make each `iostream` class MT-safe; it also provides a facility that dynamically enables and disables the locks so that the MT-safe property can be dynamically changed. The basic functionality for I/O conversion and buffer management are organized into the `unsafe_` classes; the MT-safe additions to the library are confined to the derived classes. The MT-safe version of each class contains the same protected and public member functions as the `unsafe_` base class. Each member function in the MT-safe version class acts as a wrapper that locks the object, calls the same function in the `unsafe_` base class, and unlocks the object.

**Note** – The class `streambuf` is not derived from an unsafe class. The public and protected member functions of class `streambuf` are reentrant by locking. Unlocked versions, suffixed with `_unlocked`, are also provided.

### 4.2.1 Public Conversion Routines

A set of reentrant public functions that are MT-safe have been added to the `iostream` interface. A user-specified buffer is an additional argument to each function. These functions are described as follows.

#### TABLE 4-2  Reentrant Public Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char *oct_r (char *buf, int buflen, long num, int width)</code></td>
<td>Returns a pointer to the ASCII string that represents the number in octal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td><code>char *hex_r (char *buf, int buflen, long num, int width)</code></td>
<td>Returns a pointer to the ASCII string that represents the number in hexadecimal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
</tbody>
</table>
TABLE 4-2  Reentrant Public Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *dec_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that represents the number in decimal. A width of nonzero is assumed to be the field width for formatting. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *chr_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer to the ASCII string that contains character chr. If the width is nonzero, the string contains width blanks followed by chr. The returned value is not guaranteed to point to the beginning of the user-provided buffer.</td>
</tr>
<tr>
<td>char *form_r (char *buf, int buflen, long num, int width)</td>
<td>Returns a pointer of the string formatted by sprintf, using the format string format and any remaining arguments. The buffer must have sufficient space to contain the formatted string.</td>
</tr>
</tbody>
</table>

**Caution** – The public conversion routines of the iostream library (oct, hex, dec, chr, and form) that are present to ensure compatibility with an earlier version of libc are not MT-safe.

4.2.2 Compiling and Linking with the MT-Safe libc Library

When you build an application that uses the iostream classes of the libc library to run in a multithreaded environment, compile and link the source code of the application using the -mt option. This option passes -D_REENTRANT to the preprocessor and -lthread to the linker.

**Note** – Use -mt (rather than -lthread) to link with libc and libthread. This option ensures proper linking order of the libraries. Using -lthread improperly could cause your application to work incorrectly.

Single-threaded applications that use iostream classes do not require special compiler or linker options. By default, the compiler links with the libc library.
4.2.3 MT-Safe iostream Restrictions

The restricted definition of MT-safety for the iostream library means that a number of programming idioms used with iostream are unsafe in a multithreaded environment using shared iostream objects.

4.2.3.1 Checking Error State

To be MT-safe, error checking must occur in a critical region with the I/O operation that causes the error. The following example illustrates how to check for errors:

CODE EXAMPLE 4-1 Checking Error State

```cpp
#include <iostream.h>
enum iostate { IOok, IOeof, IOfail };

iostate read_number(istream& istr, int& num)
{
    stream_locker sl(istr, stream_locker::lock_now);
    istr >> num;
    if (istr.eof()) return IOeof;
    if (istr.fail()) return IOfail;
    return IOok;
}
```

In this example, the constructor of the stream_locker object sl locks the istream object istr. The destructor of sl, called at the termination of read_number, unlocks istr.
4.2.3.2 Obtaining Characters Extracted by Last Unformatted Input Operation

To be MT-safe, the `gcount` function must be called within a thread that has exclusive use of the `istream` object for the period that includes the execution of the last input operation and `gcount` call. The following example shows a call to `gcount`:

**CODE EXAMPLE 4-2  Calling gcount**

```cpp
#include <iostream.h>
#include <rlocks.h>

void fetch_line(istream& istr, char* line, int& linecount) {
    stream_locker sl(istr, stream_locker::lock_defer);
    sl.lock(); // lock the stream istr
    istr >> line;
    linecount = istr.gcount();
    sl.unlock(); // unlock istr
    ...
}
```

In this example, the `lock` and `unlock` member functions of class `stream_locker` define a mutual exclusion region in the program.

4.2.3.3 User-Defined I/O Operations

To be MT-safe, I/O operations defined for a user-defined type that involve a specific ordering of separate operations must be locked to define a critical region. The following example shows a user-defined I/O operation:

**CODE EXAMPLE 4-3  User-Defined I/O Operations**

```cpp
#include <rlocks.h>
#include <iostream.h>

class mystream: public istream {

    // other definitions...
    int getRecord(char* name, int& id, float& gpa);
};
```
4.2.4 Performance

Using the MT-safe classes in this version of the libC library results in some amount of performance overhead, even in a single-threaded application; however, if you use the unsafe classes of libC, this overhead can be avoided.

The scope resolution operator can be used to execute member functions of the base unsafe classes; for example:

```cpp
#include <rlocks.h>
#include <iostream.h>

int mystream::getRecord(char* name, int& id, float& gpa) {
    stream_locker sl(this, stream_locker::lock_now);
    *this >> name;
    *this >> id;
    *this >> gpa;
    return this->fail() == 0;
}
```

```cpp
cout.unsafe_ostream::put('4');
```

```cpp
cin.unsafe_istream::read(buf, len);
```

**Note** – The unsafe classes cannot be safely used in multithreaded applications.
Instead of using unsafe classes, you can make the cout and cin objects unsafe and then use the normal operations. A slight performance deterioration results. The following example shows how to use unsafe cout and cin:

**CODE EXAMPLE 4-4  Disabling MT-Safety**

```cpp
#include <iostream.h>
//disable mt-safety
cout.set_safe_flag(stream_MT::unsafe_object);
//disable mt-safety
cin.set_safe_flag(stream_MT::unsafe_object);
cout.put('4');
cin.read(buf, len);
```

When an iostream object is MT-safe, mutex locking is provided to protect the object's member variables. This locking adds unnecessary overhead to an application that only executes in a single-threaded environment. To improve performance, you can dynamically switch an iostream object to and from MT-safety. The following example makes an iostream object MT-unsafe:

**CODE EXAMPLE 4-5  Switching to MT-Unsafe**

```cpp
fs.set_safe_flag(stream_MT::unsafe_object);// disable MT-safety
.... do various i/o operations
```

You can safely use an MT-unsafe stream in code where an iostream is not shared by threads; for example, in a program that has only one thread, or in a program where each iostream is private to a thread.

If you explicitly insert synchronization into the program, you can also safely use MT-unsafe iostreams in an environment where an iostream is shared by threads. The following example illustrates the technique:

**CODE EXAMPLE 4-6  Using Synchronization with MT-Unsafe Objects**

```cpp
generic_lock() ;
fs.set_safe_flag(stream_MT::unsafe_object) ;
.... do various i/o operations
 generic_unlock() ;
```
where the generic_lock and generic_unlock functions can be any synchronization mechanism that uses such primitives as mutex, semaphores, or reader/writer locks.

**Note** – The stream_locker class provided by the libC library is the preferred mechanism for this purpose.

See Section 4.6 “Object Locks” for more information.

### 4.3 Interface Changes to the *iostream* Library

This section describes the interface changes made to the *iostream* library to make it MT-Safe.

#### 4.3.1 New Classes

The following table lists the new classes added to the libC interfaces.

**CODE EXAMPLE 4-7** New Classes

```cpp
stream_MT
stream_locker
unsafe_ios
unsafe_istream
unsafe_ostream
unsafe_iostream
unsafe_fstreambase
unsafe_strstreambase
```
4.3.2 New Class Hierarchy

The following table lists the new class hierarchy added to the iostream interfaces.

CODE EXAMPLE 4-8  New Class Hierarchy

```c++
class streambuf : public stream_MT { ... };
class unsafe_ios { ... };
class ios : virtual public unsafe_ios, public stream_MT { ... };
class unsafe_fstreambase : virtual public unsafe_ios { ... };
class fstreambase : virtual public ios, public unsafe_fstreambase
    { ... };
class unsafe_strstreambase : virtual public unsafe_ios { ... };
class strstreambase : virtual public ios, public
    unsafe_strstreambase { ... };
class unsafe_istream : virtual public unsafe_ios  { ... };
class unsafe_ostream : virtual public unsafe_ios  { ... };
class istream : virtual public ios, public unsafe_istream { ... };
class ostream : virtual public ios, public unsafe_ostream { ... };
class unsafe_ios : public unsafe_istream, public unsafe_ostream
    { ... };
```

4.3.3 New Functions

The following table lists the new functions added to the iostream interfaces.

CODE EXAMPLE 4-9  New Functions

```c++
class streambuf {
public:
    int sgetc_unlocked();
    void sgetn_unlocked(char *, int);  
    int snextc_unlocked();
    int sbumpc_unlocked();
    void stossc_unlocked();
    int in_avail_unlocked();
    int sputbackc_unlocked(char);  
    int sputc_unlocked(int);
    int sputn_unlocked(const char *, int);
    int out_waiting_unlocked();
```
protected:
  char* base_unlocked();
  char* ebuf_unlocked();
  int blen_unlocked();
  char* pbase_unlocked();
  char* eback_unlocked();
  char* gptr_unlocked();
  char* egptr_unlocked();
  char* pptr_unlocked();
  void setp_unlocked(char*, char*);
  void setg_unlocked(char*, char*, char*);
  void pbump_unlocked(int);
  void gbump_unlocked(int);
  void setb_unlocked(char*, char*, int);
  int unbuffered_unlocked();
  char* eppt UnlockeD();
  void unbuffered_unlocked(int);
  int allocate_unlocked(int);}

class filebuf : public streambuf {
public:
  int is_open_unlocked();
  filebuf* close_unlocked();
  filebuf* open_unlocked(const char*, int, int =
    filebuf::openprot);
  filebuf* attach_unlocked(int);}

class strstreambuf : public streambuf {
public:
  int freeze_unlocked();
  char* str_unlocked();
};

unsafe_ostream& endl(unsafe_ostream&);
unsafe_ostream& ends(unsafe_ostream&);
unsafe_ostream& flush(unsafe_ostream&);
unsafe_istream& ws(unsafe_istream&);
unsafe_ios& dec(unsafe_ios&);
unsafe_ios& hex(unsafe_ios&);
unsafe_ios& oct(unsafe_ios&);
char* dec_r (char* buf, int buflen, long num, int width)
4.4 Global and Static Data

Global and static data in a multithreaded application are not safely shared among threads. Although threads execute independently, they share access to global and static objects within the process. If one thread modifies such a shared object, all the other threads within the process observe the change, making it difficult to maintain state over time. In C++, class objects (instances of a class) maintain state by the values in their member variables. If a class object is shared, it is vulnerable to changes made by other threads.

When a multithreaded application uses the iostream library and includes iostream.h, the standard streams—cout, cin, cerr, and clog— are, by default, defined as global shared objects. Since the iostream library is MT-safe, it protects the state of its shared objects from access or change by another thread while a member function of an iostream object is executing. However, the scope of MT-safety for an object is confined to the period in which the object’s public member function is executing. For example,

```cpp
int c;
cin.get(c);
```

gets the next character in the get buffer and updates the buffer pointer in ThreadA. However, if the next instruction in ThreadA is another get call, the libC library does not guarantee to return the next character in the sequence. It is not guaranteed because, for example, ThreadB may have also executed the get call in the intervening period between the two get calls made in ThreadA.

See Section 4.6 “Object Locks” for strategies for dealing with the problems of shared objects and multithreading.
4.5 Sequence Execution

Frequently, when `iostream` objects are used, a sequence of I/O operations must be MT-safe. For example, the code:

```cpp
    cout << " Error message:" << errstring[err_number] << "\n";
```

involves the execution of three member functions of the `cout` stream object. Since `cout` is a shared object, the sequence must be executed atomically as a critical section to work correctly in a multithreaded environment. To perform a sequence of operations on an `iostream` class object atomically, you must use some form of locking.

The `libC` library now provides the `stream_locker` class for locking operations on an `iostream` object. See Section 4.6 “Object Locks” for information about the `stream_locker` class.

4.6 Object Locks

The simplest strategy for dealing with the problems of shared objects and multithreading is to avoid the issue by ensuring that `iostream` objects are local to a thread. For example,

- Declare objects locally within a thread’s entry function.
- Declare objects in thread-specific data. (For information on how to use thread specific data, see the `thr_keycreate(3T)` man page.)
- Dedicate a stream object to a particular thread. The object thread is private by convention.

However, in many cases, such as default shared standard stream objects, it is not possible to make the objects local to a thread, and an alternative strategy is required.

To perform a sequence of operations on an `iostream` class object atomically, you must use some form of locking. Locking adds some overhead even to a single-threaded application. The decision whether to add locking or make `iostream` objects private to a thread depends on the thread model chosen for the application: Are the threads to be independent or cooperating?

- If each independent thread is to produce or consume data using its own `iostream` object, the `iostream` objects are private to their respective threads and locking is not required.
If the threads are to cooperate (that is, they are to share the same `iostream` object), then access to the shared object must be synchronized and some form of locking must be used to make sequential operations atomic.

4.6.1 Class `stream_locker`

The `iostream` library provides the `stream_locker` class for locking a series of operations on an `iostream` object. You can, therefore, minimize the performance overhead incurred by dynamically enabling or disabling locking in `iostream` objects.

Objects of class `stream_locker` can be used to make a sequence of operations on a stream object atomic. For example, the code shown in the example below seeks to find a position in a file and reads the next block of data.

**CODE EXAMPLE 4-10 Example of Using Locking Operations**

```cpp
#include <fstream.h>
#include <rlocks.h>

void lock_example (fstream& fs)
{
    const int len = 128;
    char buf[len];
    int offset = 48;
    stream_locker s_lock(fs, stream_locker::lock_now);
    . . . . . // open file
    fs.seekg(offset, ios::beg);
    fs.read(buf, len);
}
```

In this example, the constructor for the `stream_locker` object defines the beginning of a mutual exclusion region in which only one thread can execute at a time. The destructor, called after the return from the function, defines the end of the mutual exclusion region. The `stream_locker` object ensures that both the seek to a particular offset in a file and the read from the file are performed together, atomically, and that `ThreadB` cannot change the file offset before the original `ThreadA` reads the file.
An alternative way to use a stream_locker object is to explicitly define the mutual exclusion region. In the following example, to make the I/O operation and subsequent error checking atomic, lock and unlock member function calls of a vbstream_locker object are used.

**CODE EXAMPLE 4-11** Making I/O Operation and Error Checking Atomic

```cpp
{...
    stream_locker file_lck(openfile_stream,
        stream_locker::lock_defer);
    ....
    file_lck.lock();  // lock openfile_stream
    openfile_stream << "Value: " << int_value << "\n";
    if(!openfile_stream) {
        file_error("Output of value failed\n");
        return;
    }
    file_lck.unlock();  // unlock openfile_stream
}
```

For more information, see the stream_locker(3CC4) man page.

## 4.7 MT-Safe Classes

You can extend or specialize the functionality of the iostream classes by deriving new classes. If objects instantiated from the derived classes will be used in a multithreaded environment, the classes must be MT-safe.

Considerations when deriving MT-safe classes include:

- Making a class object MT-safe by protecting the internal state of the object from multiple-thread modification. To do this, serialize access to member variables in public and protected member functions with mutex locks.
- Making a sequence of calls to member functions of an MT-safe base class atomic, using a stream_locker object.
- Avoiding locking overhead by using the _unlocked member functions of streambuf within critical regions defined by stream_locker objects.
- Locking the public virtual functions of class streambuf in case the functions are called directly by an application. These functions are: xsgetn, underflow, pbackfail, xsputn, overflow, seekoff, and seekpos.
Extending the formatting state of an `ios` object by using the member functions `iword` and `pword` in class `ios`. However, a problem can occur if more than one thread is sharing the same index to an `iword` or `pword` function. To make the threads MT-safe, use an appropriate locking scheme.

- Locking member functions that return the value of a member variable greater in size than a `char`.

### 4.8 Object Destruction

Before an `iostream` object that is shared by several threads is deleted, the main thread must verify that the subthreads are finished with the shared object. The following example shows how to safely destroy a shared object.

**CODE EXAMPLE 4-12  Destroying a Shared Object**

```c
#include <fstream.h>
#include <thread.h>
fstream* fp;

void *process_rtn(void*)
{
    // body of sub-threads which uses fp...
}

multi_process(const char* filename, int numthreads)
{
    fp = new fstream(filename, ios::in); // create fstream object
    // before creating threads.

    // create threads
    for (int i=0; i<numthreads; i++)
        thr_create(0, STACKSIZE, process_rtn, 0, 0, 0);

    ... // wait for threads to finish
    for (int i=0; i<numthreads; i++)
        thr_join(0, 0, 0);

    delete fp; // delete fstream object after
    fp = NULL; // all threads have completed.
}
```
4.9 An Example Application

The following code provides an example of a multiply-threaded application that uses `iostream` objects from the libC library in an MT-safe way.

The example application creates up to 255 threads. Each thread reads a different input file, one line at a time, and outputs the line to an output file, using the standard output stream, `cout`. The output file, which is shared by all threads, is tagged with a value that indicates which thread performed the output operation.

CODE EXAMPLE 4-13 Using `iostream` Objects in an MT-Safe Way

```
// create tagged thread data
// the output file is of the form:
// <tag><string of data>\n
// where tag is an integer value in a unsigned char.
// Allows up to 255 threads to be run in this application
// <string of data> is any printable characters
// Because tag is an integer value written as char,
// you need to use od to look at the output file, suggest:
//     od -c out.file | more

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <thread.h>

struct thread_args {
    char* filename;
    int thread_tag;
};

const int thread_bufsize = 256;

// entry routine for each thread
void* ThreadDuties(void* v) {
    // obtain arguments for this thread
    thread_args* tt = (thread_args*)v;
    char ibuf[thread_bufsize];
    // open thread input file
    ifstream instr(tt->filename);
    stream_locker lockout(cout, stream_locker::lock_defer);
    while(1) {
```
// read a line at a time
instr.getline(ibuf, thread_bufsize - 1, '\n');
if(instr.eof())
    break;
// lock cout stream so the i/o operation is atomic
lockout.lock();
// tag line and send to cout
    cout << (unsigned char)tt->thread_tag << ibuf << "\n";
lockout.unlock();
}
return 0;

int main(int argc, char** argv) {
    // argv: 1+ list of filenames per thread
    if(argc < 2) {
        cout << "usage: " << argv[0] << " <files..>\n";
        exit(1);
    }
    int num_threads = argc - 1;
    int total_tags = 0;

    // array of thread_ids
    thread_t created_threads[thread_bufsize];
    // array of arguments to thread entry routine
    thread_args thr_args[thread_bufsize];
    int i;
    for (i = 0; i < num_threads; i++) {
        thr_args[i].filename = argv[1 + i];
        // assign a tag to a thread - a value less than 256
        thr_args[i].thread_tag = total_tags++;
        // create threads
        thr_create(0, 0, ThreadDuties, &thr_args[i],
            THR_SUSPENDED, &created_threads[i]);
    }

    for(i = 0; i < num_threads; i++) {
        thr_continue(created_threads[i]);
    }
    for(i = 0; i < num_threads; i++) {
        thr_join(created_threads[i], 0, 0);
    }
    return 0;
}
The C++ Standard Library

When compiling in default (standard) mode, the compiler has access to the complete library specified by the C++ standard. The library components include what is informally known as the Standard Template Library (STL), as well as the following components.

- string classes
- numeric classes
- the standard version of stream I/O classes
- basic memory allocation
- exception classes
- run-time type information

The term STL does not have a formal definition, but is usually understood to include containers, iterators, and algorithms. The following subset of the standard library headers can be thought of as comprising the STL.

- `<algorithm>`
- `<deque>`
- `<iterator>`
- `<list>`
- `<map>`
- `<memory>`
- `<queue>`
- `<set>`
- `<stack>`
- `<utility>`
- `<vector>`

The C++ standard library (libCstd) is based on the RogueWave™ Standard C++ Library, Version 2. This library is available only for the default mode (`-compat=5`) of the compiler and is not supported with use of the `-compat` or `-compat=4` options.
If you need to use your own version of the C++ standard library instead of the version that is supplied with the compiler, you can do so by specifying the 
\_library=no\%Cstd option. Replacing the standard library that is distributed with the compiler is risky, and good results are not guaranteed. For more information, see the chapter on using libraries in the C++ User’s Guide.

For details about the standard library, see the Standard C++ Library User’s Guide and the Standard C++ Class Library Reference. The “Related Documentation” section in the preface contains information about accessing this documentation. For a list of available books about the C++ standard library see “Commercially Available Books” in the preface.

5.1 C++ Standard Library Header Files

TABLE 5-1 lists the headers for the complete standard library along with a brief description of each.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;algorithm&gt;</td>
<td>Standard algorithms that operate on containers</td>
</tr>
<tr>
<td>&lt;bitset&gt;</td>
<td>Fixed-size sequences of bits</td>
</tr>
<tr>
<td>&lt;complex&gt;</td>
<td>The numeric type representing complex numbers</td>
</tr>
<tr>
<td>&lt;deque&gt;</td>
<td>Sequences supporting addition and removal at each end</td>
</tr>
<tr>
<td>&lt;exception&gt;</td>
<td>Predefined exception classes</td>
</tr>
<tr>
<td>&lt;fstream&gt;</td>
<td>Stream I/O on files</td>
</tr>
<tr>
<td>&lt;functional&gt;</td>
<td>Function objects</td>
</tr>
<tr>
<td>&lt;iomanip&gt;</td>
<td>iostream manipulators</td>
</tr>
<tr>
<td>&lt;ios&gt;</td>
<td>iostream base classes</td>
</tr>
<tr>
<td>&lt;iosfwd&gt;</td>
<td>Forward declarations of iostream classes</td>
</tr>
<tr>
<td>&lt;iostream&gt;</td>
<td>Basic stream I/O functionality</td>
</tr>
<tr>
<td>&lt;istream&gt;</td>
<td>Input I/O streams</td>
</tr>
<tr>
<td>&lt;iterator&gt;</td>
<td>Class for traversing a sequence</td>
</tr>
<tr>
<td>&lt;limits&gt;</td>
<td>Properties of numeric types</td>
</tr>
<tr>
<td>&lt;list&gt;</td>
<td>Ordered sequences</td>
</tr>
<tr>
<td>&lt;locale&gt;</td>
<td>Support for internationalization</td>
</tr>
</tbody>
</table>
5.2 C++ Standard Library Man Pages

TABLE 5-2 lists the documentation available for each of the components of the standard library.

TABLE 5-2 Man Pages for C++ Standard Library

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>Generic algorithms for performing various operations on containers and sequences</td>
</tr>
<tr>
<td>Associative Containers</td>
<td>Ordered containers</td>
</tr>
<tr>
<td>Bidirectional_Iterators</td>
<td>An iterator that can both read and write and can traverse a container in both directions</td>
</tr>
<tr>
<td>Containers</td>
<td>A standard template library (STL) collection</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Forward_Iterators</td>
<td>A forward-moving iterator that can both read and write</td>
</tr>
<tr>
<td>Function_Objects</td>
<td>Object with an <code>operator()</code> defined</td>
</tr>
<tr>
<td>Heap_Operations</td>
<td>See entries for <code>make_heap</code>, <code>pop_heap</code>, <code>push_heap</code> and <code>sort_heap</code></td>
</tr>
<tr>
<td>Input_Iterators</td>
<td>A read-only, forward moving iterator</td>
</tr>
<tr>
<td>Insert_Iterators</td>
<td>An iterator adaptor that allows an iterator to insert into a container rather than overwrite elements in the container</td>
</tr>
<tr>
<td>Iterators</td>
<td>Pointer generalizations for traversal and modification of collections</td>
</tr>
<tr>
<td>Negators</td>
<td>Function adaptors and function objects used to reverse the sense of predicate function objects</td>
</tr>
<tr>
<td>Operators</td>
<td>Operators for the C++ Standard Template Library</td>
</tr>
<tr>
<td>Output_Iterators</td>
<td>A write-only, forward moving iterator</td>
</tr>
<tr>
<td>Predicates</td>
<td>A function or a function object that returns a boolean (true/false) value or an integer value</td>
</tr>
<tr>
<td>Random_Access_Iterators</td>
<td>An iterator that reads, writes, and allows random access to a container</td>
</tr>
<tr>
<td>Sequences</td>
<td>A container that organizes a set of sequences</td>
</tr>
<tr>
<td>Stream_Iterators</td>
<td>Includes iterator capabilities for ostream and istreams that allow generic algorithms to be used directly on streams</td>
</tr>
<tr>
<td><code>__distance_type</code></td>
<td>Determines the type of distance used by an iterator—obsolete</td>
</tr>
<tr>
<td><code>__iterator_category</code></td>
<td>Determines the category to which an iterator belongs—obsolete</td>
</tr>
<tr>
<td><code>__reverse_bi_iterator</code></td>
<td>An iterator that traverses a collection backwards</td>
</tr>
<tr>
<td>accumulate</td>
<td>Accumulates all elements within a range into a single value</td>
</tr>
<tr>
<td>adjacent_difference</td>
<td>Outputs a sequence of the differences between each adjacent pair of elements in a range</td>
</tr>
<tr>
<td>adjacent_find</td>
<td>Find the first adjacent pair of elements in a sequence that are equivalent</td>
</tr>
<tr>
<td>advance</td>
<td>Moves an iterator forward or backward (if available) by a certain distance</td>
</tr>
</tbody>
</table>
### Table 5-2: Man Pages for C++ Standard Library (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocator</td>
<td>The default allocator object for storage management in Standard Library containers</td>
</tr>
<tr>
<td>auto_ptr</td>
<td>A simple, smart pointer class</td>
</tr>
<tr>
<td>back_insert_iterator</td>
<td>An insert iterator used to insert items at the end of a collection</td>
</tr>
<tr>
<td>back_inserter</td>
<td>An insert iterator used to insert items at the end of a collection</td>
</tr>
<tr>
<td>basic_filebuf</td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td>basic_fstream</td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ifstream</td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ios</td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td>basic_iostream</td>
<td>Assists in formatting and interpreting sequences of characters controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_istream</td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_iostream</td>
<td>Supports reading objects of class basic_string&lt;charT,traits,Allocator&gt; from an array in memory</td>
</tr>
<tr>
<td>basic_ofstream</td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>basic_ostream</td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>basic_ostream</td>
<td>Supports writing objects of class basic_string&lt;charT,traits,Allocator&gt;</td>
</tr>
<tr>
<td>basic_streambuf</td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td>basic_string</td>
<td>A templatized class for handling sequences of character-like entities</td>
</tr>
<tr>
<td>basic_stringbuf</td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
<tr>
<td>basic_stringstream</td>
<td>Supports writing and reading objects of class basic_string&lt;charT,traits,Allocator&gt; to or from an array in memory</td>
</tr>
</tbody>
</table>
### TABLE 5-2  Man Pages for C++ Standard Library  (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary_function</td>
<td>Base class for creating binary function objects</td>
</tr>
<tr>
<td>binary_negate</td>
<td>A function object that returns the complement of the result of its binary predicate</td>
</tr>
<tr>
<td>binary_search</td>
<td>Performs a binary search for a value on a container</td>
</tr>
<tr>
<td>bind1st</td>
<td>Templatized utilities to bind values to function objects.</td>
</tr>
<tr>
<td>bind2nd</td>
<td>Templatized utilities to bind values to function objects.</td>
</tr>
<tr>
<td>binder1st</td>
<td>Templatized utilities to bind values to function objects.</td>
</tr>
<tr>
<td>binder2nd</td>
<td>Templatized utilities to bind values to function objects.</td>
</tr>
<tr>
<td>bitset</td>
<td>A template class and related functions for storing and manipulating fixed-size sequences of bits</td>
</tr>
<tr>
<td>cerr</td>
<td>Controls output to an unbuffered stream buffer associated with the object stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>char_traits</td>
<td>A traits class with types and operations for the basic_string container and iostream classes</td>
</tr>
<tr>
<td>cin</td>
<td>Controls input from a stream buffer associated with the object stdin declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>clog</td>
<td>Controls output to a stream buffer associated with the object stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>codecvt</td>
<td>A code conversion facet</td>
</tr>
<tr>
<td>codecvt_byname</td>
<td>A facet that includes code set conversion classification facilities based on the named locales</td>
</tr>
<tr>
<td>collate</td>
<td>A string collation, comparison, and hashing facet</td>
</tr>
<tr>
<td>collate_byname</td>
<td>A string collation, comparison, and hashing facet</td>
</tr>
<tr>
<td>compare</td>
<td>A binary function or a function object that returns true or false</td>
</tr>
<tr>
<td>complex</td>
<td>C++ complex number library</td>
</tr>
<tr>
<td>copy</td>
<td>Copies a range of elements</td>
</tr>
<tr>
<td>copy_backward</td>
<td>Copies a range of elements</td>
</tr>
<tr>
<td>count</td>
<td>Count the number of elements in a container that satisfy a given condition</td>
</tr>
<tr>
<td>count_if</td>
<td>Count the number of elements in a container that satisfy a given condition</td>
</tr>
<tr>
<td>cout</td>
<td>Controls output to a stream buffer associated with the object stdout declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ctype</td>
<td>A facet that includes character classification facilities</td>
</tr>
<tr>
<td>ctype_byname</td>
<td>A facet that includes character classification facilities based on the named locales</td>
</tr>
<tr>
<td>deque</td>
<td>A sequence that supports random access iterators and efficient insertion/deletion at both beginning and end</td>
</tr>
<tr>
<td>distance</td>
<td>Computes the distance between two iterators</td>
</tr>
<tr>
<td>divides</td>
<td>Returns the result of dividing its first argument by its second</td>
</tr>
<tr>
<td>equal</td>
<td>Compares two ranges for equality</td>
</tr>
<tr>
<td>equal_range</td>
<td>Finds the largest subrange in a collection into which a given value can be inserted without violating the ordering of the collection</td>
</tr>
<tr>
<td>equal_to</td>
<td>A binary function object that returns true if its first argument equals its second</td>
</tr>
<tr>
<td>exception</td>
<td>A class that supports logic and runtime errors</td>
</tr>
<tr>
<td>facets</td>
<td>A family of classes used to encapsulate categories of locale functionality</td>
</tr>
<tr>
<td>filebuf</td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td>fill</td>
<td>Initializes a range with a given value</td>
</tr>
<tr>
<td>fill_n</td>
<td>Initializes a range with a given value</td>
</tr>
<tr>
<td>find</td>
<td>Finds an occurrence of value in a sequence</td>
</tr>
<tr>
<td>find_end</td>
<td>Finds the last occurrence of a sub-sequence in a sequence</td>
</tr>
<tr>
<td>find_first_of</td>
<td>Finds the first occurrence of any value from one sequence in another sequence</td>
</tr>
<tr>
<td>find_if</td>
<td>Finds an occurrence of a value in a sequence that satisfies a specified predicate</td>
</tr>
<tr>
<td>for_each</td>
<td>Applies a function to each element in a range</td>
</tr>
<tr>
<td>fpos</td>
<td>Maintains position information for the iostream classes</td>
</tr>
<tr>
<td>front_insert_iterator</td>
<td>An insert iterator used to insert items at the beginning of a collection</td>
</tr>
<tr>
<td>front_inserter</td>
<td>An insert iterator used to insert items at the beginning of a collection</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>fstream</td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td>generate</td>
<td>Initialize a container with values produced by a value-generator class</td>
</tr>
<tr>
<td>generate_n</td>
<td>Initialize a container with values produced by a value-generator class</td>
</tr>
<tr>
<td>get_temporary_buffer</td>
<td>Pointer based primitive for handling memory</td>
</tr>
<tr>
<td>greater</td>
<td>A binary function object that returns true if its first argument is greater than its second</td>
</tr>
<tr>
<td>greater_equal</td>
<td>A binary function object that returns true if its first argument is greater than or equal to its second</td>
</tr>
<tr>
<td>gslice</td>
<td>A numeric array class used to represent a generalized slice from an array</td>
</tr>
<tr>
<td>gslice_array</td>
<td>A numeric array class used to represent a BLAS-like slice from a valarray</td>
</tr>
<tr>
<td>has_facet</td>
<td>A function template used to determine if a locale has a given facet</td>
</tr>
<tr>
<td>ifstream</td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>includes</td>
<td>A basic set of operation for sorted sequences</td>
</tr>
<tr>
<td>indirect_array</td>
<td>A numeric array class used to represent elements selected from a valarray</td>
</tr>
<tr>
<td>inner_product</td>
<td>Computes the inner product A X B of two ranges A and B</td>
</tr>
<tr>
<td>inplace_merge</td>
<td>Merges two sorted sequences into one</td>
</tr>
<tr>
<td>insert_iterator</td>
<td>An insert iterator used to insert items into a collection rather than overwrite the collection</td>
</tr>
<tr>
<td>inserter</td>
<td>An insert iterator used to insert items into a collection rather than overwrite the collection</td>
</tr>
<tr>
<td>ios</td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td>ios_base</td>
<td>Defines member types and maintains data for classes that inherit from it</td>
</tr>
<tr>
<td>iosfwd</td>
<td>Declares the input/output library template classes and specializes them for wide and tiny characters</td>
</tr>
<tr>
<td>isalnum</td>
<td>Determines if a character is alphabetic or numeric</td>
</tr>
</tbody>
</table>
### TABLE 5-2  Man Pages for C++ Standard Library  (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalpha</td>
<td>Determines if a character is alphabetic</td>
</tr>
<tr>
<td>iscntrl</td>
<td>Determines if a character is a control character</td>
</tr>
<tr>
<td>isdigit</td>
<td>Determines if a character is a decimal digit</td>
</tr>
<tr>
<td>isgraph</td>
<td>Determines if a character is a graphic character</td>
</tr>
<tr>
<td>islower</td>
<td>Determines whether a character is lower case</td>
</tr>
<tr>
<td>isprint</td>
<td>Determines if a character is printable</td>
</tr>
<tr>
<td>ispunct</td>
<td>Determines if a character is punctuation</td>
</tr>
<tr>
<td>isspace</td>
<td>Determines if a character is a space</td>
</tr>
<tr>
<td>istream</td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>istream_iterator</td>
<td>A stream iterator that has iterator capabilities for istreams</td>
</tr>
<tr>
<td>istreambuf_iterator</td>
<td>Reads successive characters from the stream buffer for which it was constructed</td>
</tr>
<tr>
<td>istringstream</td>
<td>Supports reading objects of class basic_string&lt;CharT,traits,Allocator&gt; from an array in memory</td>
</tr>
<tr>
<td>istrstream</td>
<td>Reads characters from an array in memory</td>
</tr>
<tr>
<td>isupper</td>
<td>Determines whether a character is upper case</td>
</tr>
<tr>
<td>isxdigit</td>
<td>Determines whether a character is a hexadecimal digit</td>
</tr>
<tr>
<td>iter_swap</td>
<td>Exchanges values in two locations</td>
</tr>
<tr>
<td>iterator</td>
<td>A base iterator class</td>
</tr>
<tr>
<td>iterator_traits</td>
<td>Returns basic information about an iterator</td>
</tr>
<tr>
<td>less</td>
<td>A binary function object that returns true if its first argument is less than its second</td>
</tr>
<tr>
<td>less_equal</td>
<td>A binary function object that returns true if its first argument is less than or equal to its second</td>
</tr>
<tr>
<td>lexicographical_compare</td>
<td>Compares two ranges lexicographically</td>
</tr>
<tr>
<td>limits</td>
<td>Refer to numeric_limits</td>
</tr>
<tr>
<td>list</td>
<td>A sequence that supports bidirectional iterators</td>
</tr>
<tr>
<td>locale</td>
<td>A localization class containing a polymorphic set of facets</td>
</tr>
<tr>
<td>logical_and</td>
<td>A binary function object that returns true if both of its arguments are true</td>
</tr>
</tbody>
</table>
### TABLE 5-2  Man Pages for C++ Standard Library  (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical_not</td>
<td>A unary function object that returns true if its argument is false</td>
</tr>
<tr>
<td>logical_or</td>
<td>A binary function object that returns true if either of its arguments are true</td>
</tr>
<tr>
<td>lower_bound</td>
<td>Determines the first valid position for an element in a sorted container</td>
</tr>
<tr>
<td>make_heap</td>
<td>Creates a heap</td>
</tr>
<tr>
<td>map</td>
<td>An associative container with access to non-key values using unique keys</td>
</tr>
<tr>
<td>mask_array</td>
<td>A numeric array class that gives a masked view of a valarray</td>
</tr>
<tr>
<td>max</td>
<td>Finds and returns the maximum of a pair of values</td>
</tr>
<tr>
<td>max_element</td>
<td>Finds the maximum value in a range</td>
</tr>
<tr>
<td>mem_fun</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun1</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun_ref</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>mem_fun_ref1</td>
<td>Function objects that adapt a pointer to a member function, to take the place of a global function</td>
</tr>
<tr>
<td>merge</td>
<td>Merges two sorted sequences into a third sequence</td>
</tr>
<tr>
<td>messages</td>
<td>Messaging facets</td>
</tr>
<tr>
<td>messages_byname</td>
<td>Messaging facets</td>
</tr>
<tr>
<td>min</td>
<td>Finds and returns the minimum of a pair of values</td>
</tr>
<tr>
<td>min_element</td>
<td>Finds the minimum value in a range</td>
</tr>
<tr>
<td>minus</td>
<td>Returns the result of subtracting its second argument from its first</td>
</tr>
<tr>
<td>mismatch</td>
<td>Compares elements from two sequences and returns the first two elements that don’t match each other</td>
</tr>
<tr>
<td>modulus</td>
<td>Returns the remainder obtained by dividing the first argument by the second argument</td>
</tr>
<tr>
<td>money_get</td>
<td>Monetary formatting facet for input</td>
</tr>
<tr>
<td>money_put</td>
<td>Monetary formatting facet for output</td>
</tr>
<tr>
<td>moneypunct</td>
<td>Monetary punctuation facets</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>moneypunct_byname</td>
<td>Monetary punctuation facets</td>
</tr>
<tr>
<td>multimap</td>
<td>An associative container that gives access to non-key values using keys</td>
</tr>
<tr>
<td>multiplies</td>
<td>A binary function object that returns the result of multiplying its first and second arguments</td>
</tr>
<tr>
<td>multiset</td>
<td>An associative container that allows fast access to stored key values</td>
</tr>
<tr>
<td>negate</td>
<td>Unary function object that returns the negation of its argument</td>
</tr>
<tr>
<td>next_permutation</td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td>not1</td>
<td>A function adaptor used to reverse the sense of a unary predicate function object</td>
</tr>
<tr>
<td>not2</td>
<td>A function adaptor used to reverse the sense of a binary predicate function object</td>
</tr>
<tr>
<td>not_equal_to</td>
<td>A binary function object that returns true if its first argument is not equal to its second</td>
</tr>
<tr>
<td>nth_element</td>
<td>Rearranges a collection so that all elements lower in sorted order than the nth element come before it and all elements higher in sorter order than the nth element come after it</td>
</tr>
<tr>
<td>num_get</td>
<td>A numeric formatting facet for input</td>
</tr>
<tr>
<td>num_put</td>
<td>A numeric formatting facet for output</td>
</tr>
<tr>
<td>numeric_limits</td>
<td>A class for representing information about scalar types</td>
</tr>
<tr>
<td>numeric_limits</td>
<td>A numeric limits</td>
</tr>
<tr>
<td>numpunct</td>
<td>A numeric punctuation facet</td>
</tr>
<tr>
<td>numpunct_byname</td>
<td>A numeric punctuation facet</td>
</tr>
<tr>
<td>ofstream</td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>ostream</td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>ostream_iterator</td>
<td>Stream iterators allow for use of iterators with ostreams and istreams</td>
</tr>
<tr>
<td>ostreambuf_iterator</td>
<td>Writes successive characters onto the stream buffer object from which it was constructed</td>
</tr>
</tbody>
</table>
| ostringstream      | Supports writing objects of class basic_string<

\texttt{charT,traits,Allocator}>
<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>ostrstream</td>
<td>Writes to an array in memory</td>
</tr>
<tr>
<td>pair</td>
<td>A template for heterogeneous pairs of values</td>
</tr>
<tr>
<td>partial_sort</td>
<td>Templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>partial_sort_copy</td>
<td>Templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>partial_sum</td>
<td>Calculates successive partial sums of a range of values</td>
</tr>
<tr>
<td>partition</td>
<td>Places all of the entities that satisfy the given predicate before all of the entities that do not</td>
</tr>
<tr>
<td>permutation</td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td>plus</td>
<td>A binary function object that returns the result of adding its first and second arguments</td>
</tr>
<tr>
<td>pointer_to_binary_function</td>
<td>A function object that adapts a pointer to a binary function, to take the place of a binary_function</td>
</tr>
<tr>
<td>pointer_to_unary_function</td>
<td>A function object class that adapts a pointer to a function, to take the place of a unary_function</td>
</tr>
<tr>
<td>pop_heap</td>
<td>Moves the largest element off the heap</td>
</tr>
<tr>
<td>prev_permutation</td>
<td>Generates successive permutations of a sequence based on an ordering function</td>
</tr>
<tr>
<td>priority_queue</td>
<td>A container adapter that behaves like a priority queue</td>
</tr>
<tr>
<td>ptr_fun</td>
<td>A function that is overloaded to adapt a pointer to a function, to take the place of a function</td>
</tr>
<tr>
<td>push_heap</td>
<td>Places a new element into a heap</td>
</tr>
<tr>
<td>queue</td>
<td>A container adaptor that behaves like a queue (first in, first out)</td>
</tr>
<tr>
<td>random_shuffle</td>
<td>Randomly shuffles elements of a collection</td>
</tr>
<tr>
<td>raw_storage_iterator</td>
<td>Enables iterator-based algorithms to store results into uninitialized memory</td>
</tr>
<tr>
<td>remove</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>remove_copy</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remove_copy_if</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>remove_if</td>
<td>Moves desired elements to the front of a container, and returns an iterator that describes where the sequence of desired elements ends</td>
</tr>
<tr>
<td>replace</td>
<td>Substitutes elements in a collection with new values</td>
</tr>
<tr>
<td>replace_copy</td>
<td>Substitutes elements in a collection with new values, and moves the revised sequence into result</td>
</tr>
<tr>
<td>replace_copy_if</td>
<td>Substitutes elements in a collection with new values, and moves the revised sequence into result</td>
</tr>
<tr>
<td>replace_if</td>
<td>Substitutes elements in a collection with new values</td>
</tr>
<tr>
<td>return_temporary_buffer</td>
<td>A pointer-based primitive for handling memory</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverses the order of elements in a collection</td>
</tr>
<tr>
<td>reverse_copy</td>
<td>Reverses the order of elements in a collection while copying them to a new collection</td>
</tr>
<tr>
<td>reverse_iterator</td>
<td>An iterator that traverses a collection backwards</td>
</tr>
<tr>
<td>rotate</td>
<td>Swaps the segment that contains elements from first through middle-1 with the segment that contains the elements from middle through last</td>
</tr>
<tr>
<td>rotate_copy</td>
<td>Swaps the segment that contains elements from first through middle-1 with the segment that contains the elements from middle through last</td>
</tr>
<tr>
<td>search</td>
<td>Finds a sub-sequence within a sequence of values that is element-wise equal to the values in an indicated range</td>
</tr>
<tr>
<td>search_n</td>
<td>Finds a sub-sequence within a sequence of values that is element-wise equal to the values in an indicated range</td>
</tr>
<tr>
<td>set</td>
<td>An associative container that supports unique keys</td>
</tr>
<tr>
<td>set_difference</td>
<td>A basic set operation for constructing a sorted difference</td>
</tr>
<tr>
<td>set_intersection</td>
<td>A basic set operation for constructing a sorted intersection</td>
</tr>
<tr>
<td>set_symmetric_difference</td>
<td>A basic set operation for constructing a sorted symmetric difference</td>
</tr>
<tr>
<td>set_union</td>
<td>A basic set operation for constructing a sorted union</td>
</tr>
</tbody>
</table>
### TABLE 5-2  Man Pages for C++ Standard Library  (Continued)

<table>
<thead>
<tr>
<th>Man Page</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice</td>
<td>A numeric array class for representing a BLAS-like slice from an array</td>
</tr>
<tr>
<td>slice_array</td>
<td>A numeric array class for representing a BLAS-like slice from a valarray</td>
</tr>
<tr>
<td>smanip</td>
<td>Helper classes used to implement parameterized manipulators</td>
</tr>
<tr>
<td>smanip_fill</td>
<td>Helper classes used to implement parameterized manipulators</td>
</tr>
<tr>
<td>sort</td>
<td>A templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>sort_heap</td>
<td>Converts a heap into a sorted collection</td>
</tr>
<tr>
<td>stable_partition</td>
<td>Places all of the entities that satisfy the given predicate before all of the entities that do not, while maintaining the relative order of elements in each group</td>
</tr>
<tr>
<td>stable_sort</td>
<td>A templatized algorithm for sorting collections of entities</td>
</tr>
<tr>
<td>stack</td>
<td>A container adapter that behaves like a stack (last in, first out)</td>
</tr>
<tr>
<td>streambuf</td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td>string</td>
<td>A typedef for <code>basic_string&lt;char, char_traits&lt;char&gt;, allocator&lt;char&gt;&gt;</code></td>
</tr>
<tr>
<td>stringbuf</td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
<tr>
<td>stringstream</td>
<td>Supports writing and reading objects of class <code>basic_string&lt;charT,traits,Allocator&gt;</code> to/from an array in memory</td>
</tr>
<tr>
<td>strstream</td>
<td>Reads and writes to an array in memory</td>
</tr>
<tr>
<td>strstreambuf</td>
<td>Associates either the input sequence or the output sequence with a tiny character array whose elements store arbitrary values</td>
</tr>
<tr>
<td>swap</td>
<td>Exchanges values</td>
</tr>
<tr>
<td>swap_ranges</td>
<td>Exchanges a range of values in one location with those in another</td>
</tr>
<tr>
<td>time_get</td>
<td>A time formatting facet for input</td>
</tr>
<tr>
<td>time_get_byname</td>
<td>A time formatting facet for input, based on the named locales</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>time_put</td>
<td>A time formatting facet for output</td>
</tr>
<tr>
<td>time_put_byname</td>
<td>A time formatting facet for output, based on the named locales</td>
</tr>
<tr>
<td>tolower</td>
<td>Converts a character to lower case.</td>
</tr>
<tr>
<td>toupper</td>
<td>Converts a character to upper case</td>
</tr>
<tr>
<td>transform</td>
<td>Applies an operation to a range of values in a collection and stores the result</td>
</tr>
<tr>
<td>unary_function</td>
<td>A base class for creating unary function objects</td>
</tr>
<tr>
<td>unary_negate</td>
<td>A function object that returns the complement of the result of its unary predicate</td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td>An algorithm that uses construct to copy values from one range to another location</td>
</tr>
<tr>
<td>uninitialized_fill</td>
<td>An algorithm that uses the construct algorithm for setting values in a collection</td>
</tr>
<tr>
<td>uninitialized_fill_n</td>
<td>An algorithm that uses the construct algorithm for setting values in a collection</td>
</tr>
<tr>
<td>unique</td>
<td>Removes consecutive duplicates from a range of values and places the resulting unique values into the result</td>
</tr>
<tr>
<td>unique_copy</td>
<td>Removes consecutive duplicates from a range of values and places the resulting unique values into the result</td>
</tr>
<tr>
<td>upper_bound</td>
<td>Determines the last valid position for a value in a sorted container</td>
</tr>
<tr>
<td>use_facet</td>
<td>A template function used to obtain a facet</td>
</tr>
<tr>
<td>valarray</td>
<td>An optimized array class for numeric operations</td>
</tr>
<tr>
<td>vector</td>
<td>A sequence that supports random access iterators</td>
</tr>
<tr>
<td>wcerr</td>
<td>Controls output to an unbuffered stream buffer associated with the object stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>wcin</td>
<td>Controls input from a stream buffer associated with the object stdin declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>wclog</td>
<td>Controls output to a stream buffer associated with the object stderr declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>wcout</td>
<td>Controls output to a stream buffer associated with the object stdout declared in <code>&lt;cstdio&gt;</code></td>
</tr>
<tr>
<td>wfilebuf</td>
<td>Class that associates the input or output sequence with a file</td>
</tr>
<tr>
<td>Man Page</td>
<td>Overview</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wfstream</td>
<td>Supports reading and writing of named files or devices associated with a file descriptor</td>
</tr>
<tr>
<td>wifstream</td>
<td>Supports reading from named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>wios</td>
<td>A base class that includes the common functions required by all streams</td>
</tr>
<tr>
<td>wiostream</td>
<td>Assists in reading and interpreting input from sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>wistringstream</td>
<td>Supports reading objects of class basic_string&lt;charT,traits,Allocator&gt; from an array in memory</td>
</tr>
<tr>
<td>wofstream</td>
<td>Supports writing into named files or other devices associated with a file descriptor</td>
</tr>
<tr>
<td>wostream</td>
<td>Assists in formatting and writing output to sequences controlled by a stream buffer</td>
</tr>
<tr>
<td>wostringstream</td>
<td>Supports writing objects of class basic_string&lt;charT,traits,Allocator&gt;</td>
</tr>
<tr>
<td>wstreambuf</td>
<td>Abstract base class for deriving various stream buffers to facilitate control of character sequences</td>
</tr>
<tr>
<td>wstring</td>
<td>A typedef for basic_string&lt;wchar_t, char_traits&lt;wchar_t&gt;, allocator&lt;wchar_t&gt;&gt;</td>
</tr>
<tr>
<td>wstringbuf</td>
<td>Associates the input or output sequence with a sequence of arbitrary characters</td>
</tr>
</tbody>
</table>
Index

SYMBOLS

<table>
<thead>
<tr>
<th>Operator</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>! operator, ios::</td>
<td>3-6, 3-10</td>
</tr>
<tr>
<td>() parentheses, iostream</td>
<td>3-4</td>
</tr>
<tr>
<td>&lt;&lt; insertion operator</td>
<td>complex, 2-7</td>
</tr>
<tr>
<td></td>
<td>iostream::, 3-4, 3-5</td>
</tr>
<tr>
<td>&gt;&gt; extractor operator</td>
<td>complex, 2-7</td>
</tr>
<tr>
<td></td>
<td>iostream::, 3-7</td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th>Name</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs, complex::</td>
<td>2-4</td>
</tr>
<tr>
<td>absolute value, complex numbers</td>
<td>2-2</td>
</tr>
<tr>
<td>accumulate, man pages</td>
<td>5-4</td>
</tr>
<tr>
<td>acos, complex::</td>
<td>2-5</td>
</tr>
<tr>
<td>adjacent_difference, man pages</td>
<td>5-4</td>
</tr>
<tr>
<td>adjacent_find, man pages</td>
<td>5-4</td>
</tr>
<tr>
<td>advance, man pages</td>
<td>5-4</td>
</tr>
<tr>
<td>&lt;algorithm&gt;, header file</td>
<td>5-2</td>
</tr>
<tr>
<td>Algorithms, man pages</td>
<td>5-3</td>
</tr>
<tr>
<td>allocator, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>ang, complex::</td>
<td>2-4</td>
</tr>
<tr>
<td>angle, complex numbers</td>
<td>2-2</td>
</tr>
<tr>
<td>app, ios::</td>
<td>3-12</td>
</tr>
<tr>
<td>applications</td>
<td></td>
</tr>
<tr>
<td>linking</td>
<td>4-4</td>
</tr>
<tr>
<td>MT-safe</td>
<td>4-2</td>
</tr>
<tr>
<td>using MT-safe iostream objects</td>
<td>4-17 to 4-18</td>
</tr>
<tr>
<td>applicator, paracetabized manipulators</td>
<td>3-19</td>
</tr>
<tr>
<td>arithmetic library, complex</td>
<td>2-1 to 2-10</td>
</tr>
<tr>
<td>asin, complex::</td>
<td>2-5</td>
</tr>
<tr>
<td>assignment, iostream</td>
<td>3-15</td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>Name</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>back_insert_iterator, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>back_inserter, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>badbit, ios::</td>
<td>3-6</td>
</tr>
<tr>
<td>basic_filebuf, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_fstream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_ifstream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_ios, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_iostream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_istream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_ostream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_ostringstream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_stringbuf, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>basic_stringstream, man pages</td>
<td>5-5</td>
</tr>
<tr>
<td>beg, ios::</td>
<td>3-14</td>
</tr>
<tr>
<td>Bidirectional_Itersors, man pages</td>
<td>5-3</td>
</tr>
<tr>
<td>binary input, reading</td>
<td>3-9</td>
</tr>
<tr>
<td>binary, ios::</td>
<td>3-13</td>
</tr>
<tr>
<td>binary_function, man pages</td>
<td>5-6</td>
</tr>
<tr>
<td>binary_negate, man pages</td>
<td>5-6</td>
</tr>
<tr>
<td>binary_search, man pages</td>
<td>5-6</td>
</tr>
<tr>
<td>bind1st, man pages</td>
<td>5-6</td>
</tr>
</tbody>
</table>
bind2nd, man pages, 5-6
bind1st, man pages, 5-6
bind2nd, man pages, 5-6
bitset, header file, 5-2
bitset, man pages, 5-6
buffer
  defined, 3-24
  flushing output, 3-6

C

c (char), iostream manipulator, 3-15
C++ Standard Library
  components, 5-1 to 5-16
    RogueWave, 5-1
c_exception, definition, 2-6
cartpol, complex man pages, 2-10
cerr
  iostream::, 3-1, 4-12
    man pages, 5-6
char*, extractor, 3-8 to 3-9
char, extractor, 3-9
char_traits, man pages, 5-6
characters, reading single, 3-9
chr, iostream::, 4-4
chr_r, iostream::, 4-4
cin
  iostream::, 3-1, 4-12
    man pages, 5-6
clog
  iostream::, 4-12
    man pages, 5-6
    predefined iostreams, 3-1
close, ofstream::, 3-13
codecvt, man pages, 5-6
codecvt_byname, man pages, 5-6
collate, man pages, 5-6
collate_byname, man pages, 5-6
cmpare, man pages, 5-6
compatibility mode
  libC, 3-1, 3-3
  libcomplex, 2-1
compiling, MT-safe libC library, 4-4
complex
  complex library, 2-1 to 2-2
  constructors, 2-2 to 2-3
  efficiency, 2-9
  error handling, 2-6 to 2-7
  input/output, 2-7 to 2-8
    man pages, 2-10, 5-6
    mathematical functions, 2-4 to 2-6
    mixed-mode, 2-8 to 2-9
    operators, 2-3 to 2-4
    trigonometric functions, 2-5 to 2-6
    type complex, 2-2
  complex(), constructor, 2-2 to 2-3
  <complex>, header file, 5-2
  complex.h, complex header files, 2-2
  complex_error
    definition, 2-6
    message, 2-4
  conj, complex::;, 2-4
  conjugate, of a number, 2-2
  constructors
    complex, 2-2 to 2-3
    iostream, 3-2
Containers, man pages, 5-3
copy, man pages, 5-6
copy_backward, man pages, 5-6
copying
  files, 3-21
  stream objects, 3-15
core classes, LibC, 4-2
cos, complex::, 2-5
cosh, complex::, 2-5, 2-7
count, man pages, 5-6
count_if, man pages, 5-6
cout
  iostream::, 3-1, 3-4, 4-12
    man pages, 5-6
cplx.intro, complex man pages, 2-10
cplxerr, complex man pages, 2-10
cplxexp, complex man pages, 2-10
cplxops, complex man pages, 2-10
cplxtrig, complex man pages, 2-10
cplx::, man pages, 5-7
cplx_traits, man pages, 5-7
cur, ios::, 3-14

D

data types, complex number, 2-1 to 2-2
dec
  iostream manipulator, 3-16
  iostream::, 4-4
dec_r, iostream::, 4-4
<deque>, header file, 5-2
deque, man pages, 5-7
destruction, of shared objects, 4-16
distance, man pages, 5-7
_distance_type, man pages, 5-4
divides, man pages, 5-7
double, complex value, 2-2

E
EDOM, errno setting, 2-7
end, ios::, 3-14
end1, iostream manipulator, 3-16
ends, iostream manipulator, 3-16
environment
   multithreading, 4-1 to 4-18
eofbit, ios::, 3-6
equal, man pages, 5-7
equal_range, man pages, 5-7
equal_to, man pages, 5-7
ERANGE, errno setting, 2-7
errno, definition, 2-6 to 2-7
error bits, 3-6
error checking, MT-safety, 4-5
error handling
   complex, 2-6 to 2-7
   input, 3-10 to 3-11
error messages, complex_error, 2-4
error state, iostreams, 3-5
error, iostream::, 3-6
<exception>, header file, 5-2
exception, man pages, 5-7
exp, complex::, 2-5 to 2-7
extraction
   defined, 3-24
   operators, 3-7
extractor
   char, 3-9
   char*, 3-8 to 3-9
   user-defined iostream, 3-7 to 3-8
   whitespace, 3-10

F
facets, man pages, 5-7
failbit, ios::, 3-6
file descriptors, using, 3-13 to 3-14
filebuf
   man pages, 3-22, 5-7
   streambuf::, 4-11
files
   copying, 3-12, 3-21
   opening and closing, 3-13
   repositioning, 3-14
   using fstreams with, 3-11
fill, man pages, 5-7
fill_n, man pages, 5-7
find, man pages, 5-7
find_end, man pages, 5-7
find_first_of, man pages, 5-7
find_if, man pages, 5-7
float inserter, iostream output, 3-4
flush, iostream manipulator, 3-6, 3-16
for_each, man pages, 5-7
form, iostream::, 4-4
form_r, iostream::, 4-4
format control, iostreams, 3-15
Forward_Iterators, man pages, 5-4
fpos, man pages, 5-7
front_insert_iterator, man pages, 5-7
front_inserter, man pages, 5-7
fstream
   defined, 3-24
   iostream::, 3-2, 3-11 to 3-15
   man pages, 3-22, 5-8
<fstream>, header file, 5-2
fstream.h
   iostream header file, 3-3
   using, 3-12
Function_Objects, man pages, 5-4
<functional>, header file, 5-2
functions
   MT-safe public, 4-3
   streambuf public virtual, 4-15

G
gcFixPrematureFrees, man pages, 1-3
gcInitialize, man pages, 1-3
gcmonitor
   man pages, 1-3
   Sun WorkShop Memory Monitor daemon, 1-3
gcount, iostream::, 4-6
generate, man pages, 5-8
generate_n, man pages, 5-8
<table>
<thead>
<tr>
<th>topic</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get pointer</td>
<td>3-20</td>
</tr>
<tr>
<td>get, char extractor</td>
<td>3-9</td>
</tr>
<tr>
<td>get_temporary_buffer</td>
<td>man pages, 5-8</td>
</tr>
<tr>
<td>global data, in a multithreaded application</td>
<td>4-12</td>
</tr>
<tr>
<td>global shared objects, default</td>
<td>4-12</td>
</tr>
<tr>
<td>good, ios::</td>
<td>3-6</td>
</tr>
<tr>
<td>goodbit, ios::</td>
<td>3-6</td>
</tr>
<tr>
<td>greater, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>greater_equal, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>gslice, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>gslice_array, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td>hardfail, ios::</td>
<td>3-6</td>
</tr>
<tr>
<td>has_facet, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>header files</td>
<td></td>
</tr>
<tr>
<td>complex, 2-2, 2-9</td>
<td></td>
</tr>
<tr>
<td>function of, 1-1</td>
<td></td>
</tr>
<tr>
<td>iostream, 3-3, 3-16, 4-12</td>
<td></td>
</tr>
<tr>
<td>Standard Library, 5-2 to 5-3</td>
<td></td>
</tr>
<tr>
<td>Heap_Operations, man pages</td>
<td>5-4</td>
</tr>
<tr>
<td>hex</td>
<td></td>
</tr>
<tr>
<td>iostream manipulator, 3-16</td>
<td></td>
</tr>
<tr>
<td>iostream::</td>
<td>4-4</td>
</tr>
<tr>
<td>hex_r, iostream::</td>
<td>4-3</td>
</tr>
<tr>
<td>hierarchy, new iostream class</td>
<td>4-10</td>
</tr>
<tr>
<td>HTML, accessing man pages formatted in</td>
<td>1-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>topic</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>i (long), iostream manipulator</td>
<td>3-15</td>
</tr>
<tr>
<td>I/O library</td>
<td>3-1</td>
</tr>
<tr>
<td>ifstream</td>
<td></td>
</tr>
<tr>
<td>iostream::</td>
<td>3-2, 3-11</td>
</tr>
<tr>
<td>istream::</td>
<td>3-11</td>
</tr>
<tr>
<td>man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>imag, complex::</td>
<td>2-5</td>
</tr>
<tr>
<td>in, ios::</td>
<td>3-12</td>
</tr>
<tr>
<td>includes, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>indirect_array, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>inner_product, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>inplace_merge, man pages</td>
<td>5-8</td>
</tr>
<tr>
<td>input</td>
<td></td>
</tr>
<tr>
<td>binary, 3-9</td>
<td></td>
</tr>
<tr>
<td>error handling, 3-10 to 3-11</td>
<td></td>
</tr>
</tbody>
</table>
structure, 3-2 to 3-3
terminology, 3-24
using, 3-3
`<iostream>`, header file, 5-2
`iostream.h`, `iostream` header file, 3-3, 4-12
`isalnum`, man pages, 5-8
`isalpha`, man pages, 5-9
`iscntrl`, man pages, 5-9
`isdigit`, man pages, 5-9
`isgraph`, man pages, 5-9
`islower`, man pages, 5-9
`isprint`, man pages, 5-9
`ispunct`, man pages, 5-9
`isspace`, man pages, 5-9
`istream`
   `iosstream::`, 3-2
   man pages, 3-22, 5-9
`<iostream>`, header file, 5-2
`istream_iterator`, man pages, 5-9
`istreambuf_iterator`, man pages, 5-9
`istringstream`, man pages, 5-9
`istrstream`
   `iosstream::`, 3-2
   man pages, 5-9
`isupper`, man pages, 5-9
`isxdigit`, man pages, 5-9
`iter_swap`, man pages, 5-9
`<iterator>`, header file, 5-2
`iterator`, man pages, 5-9
   `_iterator_category`, man pages, 5-4
   `iterator_traits`, man pages, 5-9
`Iterators`, man pages, 5-4
`iword`, `ios::`, 4-16

L
left-shift operator
   complex, 2-7
   `iostream`, 3-4
`less`, man pages, 5-9
`less_equal`, man pages, 5-9
`lexicographical_compare`, man pages, 5-9
libC
   compatibility mode, 3-1, 3-3
   compiling and linking MT-safety, 4-4
   core classes, 4-2
   multithreading environment library, 4-1
   new classes, 4-9
   `libcomplex`, compatibility mode, 2-1
   `libCstd`, standard mode, 2-1, 3-1
   `libgc`, Sun WorkShop Memory Monitor
      library, 1-3
   `libgc_dbg`, Sun WorkShop Memory Monitor
      library, 1-3
   `libiostream`
      multithreading environment library, 4-1
      standard mode, 3-1, 3-3
libraries
   C++ Standard, 5-1 to 5-16
   classic `iostream`, 3-1 to 3-24
   complex arithmetic, 2-1 to 2-10
   contents of a C++, 1-1
   `libC`, 4-1
   `libiostream` multithreading environment, 4-1
   Sun Workshop Memory Monitor, 1-3
   `Tools.h++`, 1-2
`<limits>`, header file, 5-2
`limits`, man pages, 5-9
linking
   `-mt` option, 4-4
   MT-safe `libC` library, 4-4
`<list>`, header file, 5-2
`list`, man pages, 5-9
`<locale>`, header file, 5-2
`locale`, man pages, 5-9
locking
   See also `stream_locker`
      mutex, 4-8, 4-15
      object, 4-13 to 4-15
      `streambuf`, 4-3
`log`, `complex::`, 2-5 to 2-7
`log10`, `complex::`, 2-5 to 2-7
`logical_and`, man pages, 5-9
`logical_not`, man pages, 5-10
`logical_or`, man pages, 5-10
`lower_bound`, man pages, 5-10
   `-lthread`, linking applications, 4-4

M
magnitude, complex numbers, 2-2
make_heap, man pages, 5-10
man pages
   accessing, 1-1 to 1-3
   C++ Standard Library, 5-3 to 5-16
      complex, 2-10

iostream, 3-1, 3-12, 3-15, 3-19 to 3-23
Memory Monitor, 1-3
manip
ios::, 3-19
man pages, 3-22
manipulators
iostreams, 3-15 to 3-19
plain, 3-17
predefined, 3-16
</map>, header file, 5-3
map, man pages, 5-10
mask_array, man pages, 5-10
math.h, complex header files, 2-9
mathematical functions, complex arithmetic
library, 2-4 to 2-6
max, man pages, 5-10
max_element, man pages, 5-10
mem_fun, man pages, 5-10
mem_fun1, man pages, 5-10
mem_fun_ref, man pages, 5-10
mem_fun_ref1, man pages, 5-10
Memory Monitor, components, 1-2
</memory>, header file, 5-3
merge, man pages, 5-10
messages, man pages, 5-10
messages_byname, man pages, 5-10
min, man pages, 5-10
min_element, man pages, 5-10
minus, man pages, 5-10
mismatch, man pages, 5-10
mixed-mode, complex arithmetic library, 2-8 to 2-9
modifications, iostream library, 4-1
modulus, man pages, 5-10
money_get, man pages, 5-10
money_put, man pages, 5-10
moneypunct, man pages, 5-10
moneypunct_byname, man pages, 5-11
-nmt, iostream::, 4-4
MT-safe
applications, 4-2
classes, considerations for deriving, 4-15
library, 4-1
object, 4-2
performance overhead, 4-7 to 4-9
public functions, 4-3
multimap, man pages, 5-11
multiplies, man pages, 5-11
multiset, man pages, 5-11
multithreading, environment, 4-1 to 4-18
mutex locks, MT-safe classes, 4-8, 4-15
mutual exclusion region, defining a, 4-15

N
n (int), iostream manipulator, 3-15
negate, man pages, 5-11
Negators, man pages, 5-4
</new>, header file, 5-3
next_permutation, man pages, 5-11
nocreate, ios::, 3-12
noreplace, ios::, 3-12
norm, complex::, 2-5
not_equal_to, man pages, 5-11
not1, man pages, 5-11
not2, man pages, 5-11
nth_element, man pages, 5-11
num_get, man pages, 5-11
num_put, man pages, 5-11
numbers, complex, 2-1 to 2-4
</numeric>, header file, 5-3
numeric_limits, man pages, 5-11
numpunct, man pages, 5-11
numpunct_byname, man pages, 5-11

O
object library, function of, 1-1
object thread, private, 4-13
objects
destruction of shared, 4-16
global shared, 4-12
strategies for dealing with shared, 4-13
stream_locker, 4-15
oct
iostream manipulator, 3-16
iostream::, 4-4
cube, iostream::, 4-3
ofstream
iostream::, 3-2
man pages, 5-11
ostream::, 3-11
omnip_manip_long, iostream::, 3-19
open, fstream::, 3-13
open_mode, ios::, 3-12 to 3-13
operations, performing a sequence of, 4-13
operators
  basic arithmetic, 2-3 to 2-4
  complex, 2-7
  iostream, 3-4, 3-5, 3-7 to 3-8
  scope resolution, 4-7
Operators, man pages, 5-4
ostream
  iostream::, 3-2
  man pages, 3-22, 5-11
<iostream>, header file, 5-3
ostream_iterator, man pages, 5-11
ostreambuf_iterator, man pages, 5-11
ostreamstream, man pages, 5-11
ostreamstream
  iostream::, 3-2
  man pages, 5-12
out, ios::, 3-12
output, 3-1
  binary, 3-7
  buffer flushing, 3-6
  cout, 3-4
  flushing, 3-6
  handling errors, 3-5
  string, 3-5 to 3-8
Output_Iterators, man pages, 5-4
overflow, streambuf::, 4-15
overhead, MT-safe class performance, 4-7 to 4-9
overloading, rules, 2-9

P
pair, man pages, 5-12
parameterized manipulators, iostreams, 3-15, 3-18 to 3-19
partial_sort, man pages, 5-12
partial_sort_copy, man pages, 5-12
partial_sum, man pages, 5-12
partition, man pages, 5-12
pbackfail, streambuf::, 4-15
peek, istream::, 3-9
peeking at input, 3-9
performance, overhead of MT-safe classes, 4-7 to 4-9
permutation, man pages, 5-12
plain manipulators, iostreams, 3-17 to 3-18
plus, man pages, 5-12
pointer_to_binary_function, man pages, 5-12
pointer_to_unary_function, man pages, 5-12
polar, complex number, 2-2
polar, complex::, 2-3, 2-5
pop_heap, man pages, 5-12
pow, complex::, 2-5
precedence, problems of, 3-4
predefined manipulators, iomanip.h, 3-16
Predicates, man pages, 5-4
prev_permutation, man pages, 5-12
priority_queue, man pages, 5-12
private, object thread, 4-13
ptr_fun, man pages, 5-12
public functions, MT-safe, 4-3
push_heap, man pages, 5-12
put pointer, streambuf, 3-20
pword, ios::, 4-16

Q
<queue>, header file, 5-3
queue, man pages, 5-12

R
Random_Access_Iterators, man pages, 5-4
random_shuffle, man pages, 5-12
raw_storage_iterator, man pages, 5-12
rdbuf, iostream::, 3-21
read, istream::, 3-9
real numbers, complex, 2-1, 2-4
real, complex::, 2-5
references
  C++, 5-2
    Tools.h++, 1-2
remove, man pages, 5-12
remove_copy, man pages, 5-12
remove_copy_if, man pages, 5-13
remove_if, man pages, 5-13
replace, man pages, 5-13
replace_copy, man pages, 5-13
replace_copy_if, man pages, 5-13
replace_if, man pages, 5-13
repositioning, within a file, 3-14 to 3-15
resetiosflags, iostream manipulator, 3-16
restrictions, MT-safe iostream, 4-5
return_temporary_buffer, man pages, 5-13
reverse, man pages, 5-13
iostream::, 3-2, 3-20, 4-2
  locking, 4-3
  man pages, 5-14
  new functions, ?? to 4-11
  public virtual functions, 4-15
  put pointer, 3-20
  queue-like, 3-21
  using, 3-21
<stringbuf>, header file, 5-3
<streampos>, 3-14
string
  iostream::, 3-5 to 3-8
  man pages, 5-14
<string>, header file, 5-3
stringbuf:, man pages, 5-14
stringstream, man pages, 5-14
strstream
  defined, 3-24
  iostream::, 3-2, 3-20
  man pages, 3-23, 5-14
streambuf.h, iostream header file, 3-3
<sstreambuf
  man pages, 5-14
  streambuf::, 4-11
swap, man pages, 5-14
swap_ranges, man pages, 5-14
sync_with_stdio, ios::, 3-11
  T
  tan, complex::, 2-6
tanh, complex::, 2-6
tellg, iostream::, 3-14
tellp, ostream::, 3-14
templates, Standard Template Library (STL), 5-1
thr_keycreate, man pages, 4-13
time_get, man pages, 5-14
time_get_byname, man pages, 5-14
time_put, man pages, 5-15
time_put_byname, man pages, 5-15
tolower, man pages, 5-15
Tools.h++ library, RogueWave, 1-2
toupper, man pages, 5-15
transform, man pages, 5-15
trigonometric functions, complex arithmetic
  library, 2-5 to 2-6
trunc, ios::, 3-12
<typeinfo>, header file, 5-3
  U
  unary_function, man pages, 5-15
  unary_negate, man pages, 5-15
  underflow, streambuf::, 4-15
  uninitialized_copy, man pages, 5-15
  uninitialized_fill, man pages, 5-15
  uninitialized_fill_n, man pages, 5-15
  unique, man pages, 5-15
  unique_copy, man pages, 5-15
  unsafe_fstream, iostream::, 4-9
  unsafe_ios, iostream::, 4-2, 4-9
  unsafe_iostream, iostream::, 4-2, 4-9
  unsafe_istream, istream::, 4-2, 4-9
  unsafe_ostream, ostream::, 4-2, 4-9
  unsafe_strstream, iostream::, 4-9
  upper_bound, man pages, 5-15
  use_facet, man pages, 5-15
  user-defined types
    iostream, 3-4
    MT-safe, 4-6 to 4-7
<utility>, header file, 5-3
  V
<valarray>, header file, 5-3
valarray, man pages, 5-15
values
  double, 2-2
  float, 3-4
  flush, 3-6
  inserting on cout, 3-4
  long, 3-19
  manipulator, 3-3, 3-15, 3-19
<vector>, header file, 5-3
vector, man pages, 5-15
void *(), ios::, 3-6
  W
wcerr, man pages, 5-15
wcin, man pages, 5-15
wclog, man pages, 5-15
wcout, man pages, 5-15
wfilebuf, man pages, 5-15
whitespace
  extractors, 3-10
leading, 3-9
  skipping, 3-10, 3-18
wifstream, man pages, 5-16
wios, man pages, 5-16
wistream, man pages, 5-16
wistringstream, man pages, 5-16
wofstream, man pages, 5-16
wostream, man pages, 5-16
wostringstream, man pages, 5-16
write, ostream::, 3-7
ws, iostream manipulator, 3-10, 3-16
wstream, man pages, 5-16
wstreambuf, man pages, 5-16
wstring, man pages, 5-16
wstringbuf, man pages, 5-16

X
X inserter, iostream, 3-4
xsgetn, streambuf::, 4-15
xsputn, streambuf::, 4-15