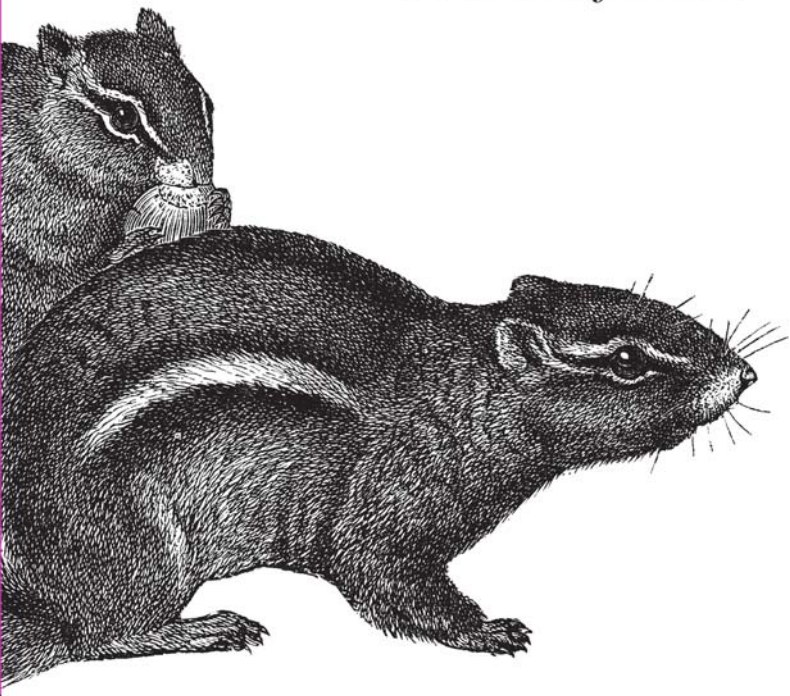
Containers, Iterators, and Algorithms

STL

Pocket Reference



O'REILLY®

Ray Lischner

C/C++

O'REILLY®

STL Pocket Reference

Do you use `search()` or `find()` to look for a value in a range? What are the arguments to `list::splice`? When do you call `mem_fun` and when do you call `mem_fun_ref`? If you're like many, you have trouble remembering these details even if you use the C++ standard template library (STL) on a daily basis. Ray Lischner's *STL Pocket Reference* comes to your rescue as a handy memory-aid that answers all your questions concisely.

The *STL Pocket Reference* documents the interface to the containers, iterators, algorithms, and function objects in the C++ STL. You'll find details of function invocations, return types, template parameters, and more.

Together with its companion volume, the *C++ Pocket Reference*, this book is sure to be a timesaver. You'll want it on your desk so you can refer to it often.

"STL is concentrated wisdom and experience, and this book is concentrated STL. No, you can't borrow my copy...buy this one!"

—Andrew Duncan
Senior Software Engineer, Expertcity, Inc.

ISBN 0-596-00556-3
US $9.95
CAN $14.95
90000
9 780596 005566
6 36920 00556 8

STL
Pocket Reference

Ray Lischner

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

STL Pocket Reference
by Ray Lischner

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Jonathan Gennick
Production Editor: Marlowe Shaeffer
Cover Designer: Ellie Volckhausen
Interior Designer: David Futato

Printing History:
October 2003: First Edition.

0-596-00556-3
[C] [6/05]

Contents

STL Pocket Reference

Introduction

STL Pocket Reference is a quick reference to the Standard Template Library portion of the standard C++ library, as documented in ISO/IEC 14882:2003(E), *Programming Languages—C++*, which is the original 1998 standard plus the corrections and additions in Technical Corrigendum 1.

The name Standard Template Library (STL) does not appear in the C++ standard, and the STL is not a formal part of the standard. Instead, the term STL usually refers to that part of the standard library that owes its heritage to the original STL, written by Alexander Stepanov, David Musser, and Meng Lee of Hewlett-Packard in the early 1990s, based on earlier work in Ada. Their innovative library of generic algorithms, iterators, and containers was quickly adopted as part of the C++ standard. The templates that were once the STL evolved and changed as the standard moved toward completion in 1998. Some other parts of the library, such as the `basic_string` class template, were changed to be more container-like, even though they were not part of the original STL.

This book describes what the STL has become: the algorithms, iterators, and containers in the standard C++ library, plus other relevant bits and pieces. For example, `bitset<>` is not a standard container, but it is included because the standard lists it in the same section as the containers and because it is useful.

The last section in this book discusses the Boost project, which is not part of the C++ standard library but is important for all C++ users. It includes several libraries that extend and enhance the STL.

The purpose of this book, as with any Pocket Reference, is to remind you of what you already know. That is, you should already know how to use algorithms, iterators, and containers. This book is a concise reference to remind you of the specific functions, classes, templates, and member functions that make up the STL. If you need a detailed reference, see *C++ in a Nutshell*. If you want to learn C++ or the STL, visit the book's web site (*http://www.tempest-sw.com/cpp*) for information about other books that are suitable for instruction.

This book is a companion to the *C++ Pocket Reference*, which covers the C++ language, but not the library. You might also want to consult the *C Pocket Reference*, which includes information about the C standard library, part of the C++ standard library.

In order to keep this book down to Pocket Reference size, it does not cover C++ I/O streams, numerics, or other parts of the C++ standard library. Nor does it offer much advice on the proper use of the STL. See the book's web site (*http://www.tempest-sw.com/cpp*) for other book recommendations.

Conventions Used in This Book

This book uses the following typographic conventions:

Italic
: Used for filenames, URLs, emphasis, and for the first use of a technical term.

`Constant Width`
: Used for identifiers and symbols.

Constant Width Italic
: Used for names that are not specified by the standard, such as hidden names of temporary objects.

Constant Width Bold
: Used in complex declarations to highlight the name being declared. In some C++ declarations, especially for templates, the name gets buried in the middle of the declaration and can be hard to spot.

In this book, the mathematical notation of [*first*, *last*) is often used to denote a range. The square bracket marks an inclusive endpoint of a range, and the parenthesis marks an exclusive endpoint of a range. Thus, [*first*, *last*) means a range that extends from *first* to *last*, including *first* but excluding *last*. Read more about ranges in the "Iterators" section later in this book.

Every function, class, and template in the STL, like the rest of the standard library, resides in the `std` namespace. This book omits the `std::` qualifier from declarations, descriptions, and most examples for the sake of brevity.

Acknowledgments

I thank my editor, Jonathan Gennick, and my technical reviewers, Reed Hedges and Andrew Duncan. I could not have written this book without the support and understanding of my wife, Cheryl, and son, Arthur.

Containers

The C++ library has a basic suite of container types (deques, lists, maps, sets, and vectors), which are described in this section. This section also discusses the `basic_string` class template because it is like a container. The non-container `bitset` template is covered in the later section, "Miscellaneous." The fundamental purpose of a container is to store multiple

objects in a single container object. Different kinds of containers have different characteristics: speed, size, and ease of use. The choice of container depends on the characteristics and behavior you require.

To add items to a container, call an `insert` member function. You can also use `push_back` or `push_front`, if they are available. Some containers offer additional means of adding items, such as `map::operator[]`.

To remove items from a container, call an `erase` member function, or a specialized version such as `pop_back` or `pop_front`. Some containers offer additional means of removing items, such as `list::remove`.

> **NOTE**
>
> Note that the standard algorithms (see "Algorithms") cannot erase items from a container. Instead, the `remove` and related algorithms reorganize the elements of a sequence in preparation for calling `erase`.

Standard Containers

The standard containers fall into two categories: sequence and associative containers. A sequence container preserves the original order in which items were added to the container. An associative container keeps items in ascending order (you can define the order relation) to speed up searching.

Sequence containers

The sequence containers are:

`basic_string`
`string`
`wstring`
: These containers represent character strings. The string class templates are not usually considered standard containers, but they meet most of the requirements of a sequence container. The header is `<string>`.

`deque`
: A deque (double-ended queue) supports fast (constant complexity) insertions and deletions at the beginning and end of the container. Inserting or deleting at any other position is slower (linear complexity), but random access to any item is fast. Items are not stored contiguously. The header is `<deque>`.

`list`
: A list supports rapid insertion or deletion at any position but does not support random access. Items are not stored contiguously. The header is `<list>`.

`vector`
: A vector is like an array, except that the vector can grow as needed. Items can be rapidly added or removed only at the end. At other positions, inserting and deleting items is slower. Random access to any item is fast. Items are stored contiguously. The header is `<vector>`.

Associative containers

The associative containers are:

`map`
`multimap`
: A map (or dictionary) is an associative container that stores pairs of keys and associated objects. Pairs are stored in ascending order of keys. A `map` requires unique keys. A `multimap` permits duplicate keys. The header for `map` and `multimap` is `<map>`.

`set`
`multiset`
: A set is an associative container that stores keys in ascending order. A `set` requires unique keys. A `multiset` permits duplicate keys. The header for `set` and `multiset` is `<set>`.

The set and map containers perform insertions, deletions, and searches with logarithmic complexity.