The Definitive Guide to Pylons

ISBN-10 (pbk): 1-59059-934-9

ISBN-13 (pbk): 978-1-59059-934-1

ISBN-13 (electronic): 978-1-4302-0534-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Steve Anglin, Matt Wade
Technical Reviewer: Michael Orr
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Beth Christmas
Copy Editor: Kim Wimpsett
Associate Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Lisa Hamilton
Indexer: Becky Hornyak
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

EXPLORATION 1

Honing Your Tools

Before you begin your exploration of the C++ landscape, you need to gather some basic supplies: a text editor, a C++ compiler, a linker, and a debugger. You can acquire these tools separately or bundled, possibly as a package deal with an integrated development environment (IDE). Options abound regardless of your platform, operating system, and budget.

If you are taking a class, the teacher will provide the tools or dictate which tools to use. If you are working at an organization that already uses C++, you probably want to use their tools, so you can become familiar with them and their proper use. If you need to acquire your own tools, check out this book's web site, `http://cpphelp.com/exploring/`. Tool versions and quality change too rapidly to provide details in print form, so you can find up-to-date suggestions on the web site. The following section gives some general advice.

Ray's Recommendations

C++ is one of the most widely used programming languages in the world: it is second only to C (depending on how you measure "widely used"). Therefore, C++ tools abound for many hardware and software environments and at a wide variety of price points.

You can choose command-line tools, which are especially popular in UNIX and UNIX-like environments, or you can opt for an IDE, which bundles all the tools into a single graphical user interface (GUI). Choose whichever style you find most comfortable. Your programs won't care what tools you use to edit, compile, and link them.

Microsoft Windows

If you are working with Microsoft Windows, I recommend Microsoft's Visual Studio (be sure that you have a current release). In particular, the venerable Visual C++ 6.0 is obsolete and out-of-date. As I write this, the current release is Visual Studio 2008. If you want a no-cost option, download Visual C++ Express from Microsoft's web site (find a current link at `http://cpphelp.com`), or for an open source solution, download MinGW, which is a port of the popular GNU compiler to Windows.

Note that C++/CLI is not the same as C++. It is a new language that Microsoft invented to help integrate C++ into the .NET environment. That's why it chose a name that incorporates C++, just as the name C++ derives from the name C. It is, however, a distinct language from C. This book covers standard C++ and nothing else. If you decide to use Visual Studio, take care that you work with C++, not C++/CLI or Managed C++ (the predecessor to C++/CLI).

Visual Studio includes a number of doodads, froufrous, and whatnots that are unimportant for your core task of learning C++. Perhaps you will need to use ATL, MFC, or .NET for your job, but for now, you can ignore all that. All you need is the C++ compiler and standard library.

If you prefer a free (as in speech) solution, the GNU compiler collection is available on Windows. Choose the Cygwin distribution, which includes a nearly complete UNIX-like environment, or MinGW, which is much smaller and might be easier to manage. In both cases, you get a good C++ compiler and library. This book's web site has links with helpful hints on installing and using these tools.

Macintosh OS 9 and Earlier

If you are stuck using an old Macintosh, download the no-cost Macintosh Programmer's Workbench from Apple's web site (link at `http://cpphelp.com`).

Everyone Else

I recommend the GNU compiler collection (GCC). The C++ compiler is called g++. Linux and BSD distributions typically come with GCC, but you might need to install the necessary developer packages. Be sure you have a recent release (version 3.4 or later) of GCC.

Mac OS X also uses GCC. For a no-cost IDE, download Xcode from Apple's web site (link at `http://cpphelp.com`).

Some hardware vendors (Sun, HP, etc.) offer a commercial compiler specifically for their hardware. This compiler might offer better optimization than GCC, but might not conform to the C++ standard as well as GCC. At least while you work through the exercises in this book, I recommend GCC. If you already have the vendor's compiler installed and you don't want to bother installing yet another compiler, go ahead and use the vendor's compiler. However, if it ever trips over an example in this book, be prepared to install GCC.

If you are using an Intel hardware platform, Intel's compiler is excellent and available at no cost for noncommercial use. Visit the book's web site for a current link.

If you want to use an IDE, choose from Eclipse, KDevelop, Anjuta, and others (go to `http://cpphelp.com` for an up-to-date list).

Read the Documentation

Now that you have your tools, take some time to read the product documentation—especially the Getting Started section. Really, I mean it. Look for tutorials and other quick introductions that will help you get up to speed with your tools. If you are using an IDE, you especially need to know how to create simple command-line projects.

IDEs typically require you to create a project, workspace, or some other envelope, or wrapper, before you can actually write a C++ program. You need to know how to do this, and I can't help you because every IDE is different. If you have a choice of project templates, choose "console," "command-line," "terminal," "C++ Tool," or some project with a similar name.

How long did it take you to read the documentation for your compiler and other tools?

_____

Was that too much time, too little time, or just right? _____

The C++ language is subject to an international standard. Every compiler (more or less) adheres to that standard, but also throws in some nonstandard extras. These extras can be useful—even necessary—for certain projects, but for this book, you need to make sure you use only standard C++. Most compilers can turn off their extensions. Even if you didn't read the documentation before, do so now to find out which options you need to enable to compile standard C++ and only standard C++.

Write down the options, for future reference.

_____

_____

_____

You may have missed some of the options; they can be obscure. To help you, Table 1-1 lists the command-line compiler options you need for Microsoft Visual C++ and for g++. This book's web site has suggestions for some other popular compilers. If you are using an IDE, look through the project options or properties to find the equivalents.

Table 1-1. *Compiler Options for Standard C++*

Compiler	Options
Visual C++ command line	`/EHsc /Za`
Visual C++ IDE	Enable C++ exceptions, disable language extensions
g++	`-pedantic -ansi`

Your First Program

Now that you have your tools, it's time to start. Fire up your favorite text editor or your C++ IDE and start your first project or create a new file. Name this file `list0101.cpp`, which is short for Listing 1-1. Several different file name extensions are popular for C++ programs. I like to use `.cpp`, where the "p" means "plus." Other common extensions are `.cxx` and `.cc`. Some compilers recognize `.C` (uppercase C) as a C++ file extension, but I don't recommend using it because it is too easy to confuse with `.c` (lowercase c), the default extension for C programs. Many desktop environments do not distinguish between uppercase and lowercase file names, further compounding the problem. Pick your favorite and stick with it. Type in the text contained within Listing 1-1. (With one exception, you can download all the code listings from this book's web site. Listing 1-1 is that exception. I want you to get used to typing C++ code in your text editor.)

Listing 1-1. *Your First C++ Program*

```
/// Sort the standard input alphabetically.
/// Read lines of text, sort them, and print the results to the standard output.
/// If the command line names a file, read from that file. Otherwise, read from
/// the standard input. The entire input is stored in memory, so don't try
/// this with input files that exceed available RAM.
```

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <iterator>
#include <ostream>
#include <string>
#include <vector>

void read(std::istream& in, std::vector<std::string>& text)
{
   std::string line;
   while (std::getline(in, line))
       text.push_back(line);
}

int main(int argc, char* argv[])
{
   // Part 1. Read the entire input into text. If the command line names a file,
   // read that file. Otherwise, read the standard input.
   std::vector<std::string> text; ///< Store the lines of text here
   if (argc < 2)
     read(std::cin, text);
   else
   {
      std::ifstream in(argv[1]);
      if (not in)
      {
         std::perror(argv[1]);
         return EXIT_FAILURE;
      }
      read(in, text);
   }

   // Part 2. Sort the text.
   std::sort(text.begin(), text.end());

   // Part 3. Print the sorted text.
   std::copy(text.begin(), text.end(),
             std::ostream_iterator<std::string>(std::cout, "\n"));
}
```

No doubt, some of this code is gibberish to you. That's okay. The point of this exercise is not to understand C++, but to make sure you can use your tools properly. The comments describe the program, which is a simple sort utility. I could have started with a trivial, "Hello, world" type of program, but that touches only a tiny fraction of the language and library. This program, being slightly more complex, does a better job at revealing possible installation or other problems with your tools.

Now go back and double-check your source code. Make sure you entered everything correctly.

Did you actually double-check the program? _____

Did you find any typos that needed correcting? _____

To err is human, and there is no shame in typographical errors. We all make them. Go back and recheck your program.

Now compile your program. If you are using an IDE, find the Compile or Build button or menu item. If you are using command-line tools, be sure to link the program, too. For historical (or hysterical) reasons, UNIX tools such as g++ typically produce an executable program named `a.out`. You should rename it to something more useful, or use the `-o` option to name an output file. Table 1-2 shows sample command lines to use for Visual C++ and g++.

Table 1-2. *Sample Command Lines to Compiler `list0101.cpp`*

Compiler	Command Line
Visual C++	`cl /EHsc /Za list0101.cpp`
g++	`g++ -o list0101 -pedantic -ansi list0101.cpp`

If you get any errors from the compiler, it means you made a mistake entering the source code; the compiler, linker, or C++ library has not been installed correctly; or the compiler, linker, or library does not conform to the C++ standard and so are unsuitable for use with this book. Triple-check you entered the text correctly. If you are confident that the error lies with the tools and not with you, check the date of publication. If the tools predate 1998, discard them immediately. They predate the standard and therefore, by definition, they cannot conform to the standard. In fact, the quality of C++ tools has improved tremendously in the last few years; so much so that I recommend discarding any tools that predate 2005. If the tools are recent, you might try the old trick of reinstalling them.

If all else fails, try a different set of tools. Download the current release of GCC or Visual Studio Express. You may need to use these tools for this book, even if you must revert to some crusty, rusty, old tools for your job.

Successful compilation is one thing, but successful execution is another. How you invoke the program depends on the operating system. In a GUI environment, you will need a console or terminal window where you can type a command line. You may need to type the complete path to the executable file, or just the program name—again, this depends on your operating system. When you run the program, it reads numbers from the standard input stream, which means whatever you type, the program reads. You then need to notify the program you are done by pressing the magic keystrokes that signal end-of-file. On most UNIX-like operating systems, press Control+D. On Windows, press Control+Z.

Running a console application from an IDE is sometimes tricky. If you aren't careful, the IDE might close the program's window before you have a chance to see any of its output. You need to ensure that the window remains visible. Some IDEs (such as Visual Studio and KDevelop) do this for you automatically, asking you to press a final Enter key before it closes the window.

If the IDE doesn't keep the window open automatically, and you can't find any option or setting to keep the window open, you can force the issue by setting a breakpoint on the

program's closing curly brace or the nearest statement where the debugger will let you set a breakpoint.

Knowing how to run the program is one thing; another is to know what numbers to type so you can test the program effectively. **How would you test `list0101` to ensure it is running correctly?**

_____

_____

_____

_____

_____

Okay, do it. **Does the program run correctly?** _____

There, that was easy, wasn't it? You should try several different sequences. Run the program with no input at all. Try it with one number. Try it with two that are already in order and two numbers that are in reverse order. Try a lot of numbers, in order. Try a lot of numbers in random order. Try a lot of numbers in reverse order.

Before you finish this Exploration, I have one more exercise. This time, the source file is more complicated. It was written by a professional stunt programmer. Do not attempt to read this program, even with adult supervision. Don't try to make any sense of the program. Above all, don't emulate the programming style used in this program. This exercise is not for you, but for your tools. Its purpose is to see whether your compiler can correctly compile this program and that your library implementation has the necessary parts of the standard library. It's not a severe torture test for a compiler, but it does touch on a few advanced C++ features.

So don't even bother trying to read the code. Just download the file `list0102.cpp` from the book's web site and try to compile and link it with your tools. (I include the full text of the program only for readers who lack convenient Internet access.) If your compiler cannot compile and run Listing 1-2 correctly, you need to replace it (your compiler, not the program). You may be able to squeak by in the early lessons, but by the end of the book, you will be writing some fairly complicated programs, and you need a compiler that is up to the task. (By the way, Listing 1-2 pretty much does the same thing as Listing 1-1.)

Listing 1-2. *Testing Your Compiler*

```
/// Sort the standard input alphabetically.
/// Read lines of text, sort them, and print the results to the standard output.
/// If the command line names a file, read from that file. Otherwise, read from
/// the standard input. The entire input is stored in memory, so don't try
/// this with input files that exceed available RAM.
///
/// Comparison uses a locale named on the command line, or the default, unnamed
/// locale if no locale is named on the command line.

#include <algorithm>
#include <cstdlib>
#include <fstream>
#include <iostream>
```

```
#include <iterator>
#include <locale>
#include <ostream>
#include <string>
#include <vector>

/// Read lines of text from @p in to @p iter. Lines are appended to @p iter.
/// @param in the input stream
/// @param iter an output iterator
template<class Ch, class Tr, class OutIter>
void read(std::basic_istream<Ch,Tr>& in, OutIter iter)
{
    std::basic_string<Ch,Tr> line;
    while (std::getline(in, line))
    {
        *iter = line;
        ++iter;
    }
}

/// Sorter function object.
/// Parameterize the class with the character type used for strings.
/// The sorter object caches a locale and its @c collate facet, to use for
/// comparing strings.
template<typename Ch>
class sorter
{
public:
    /// Construct the sorter object, caching the locale that has the given name.
    /// @param locname The name of the locale to cache.
    sorter(Ch const* locname) :
        loc_(std::locale(locname)),
        collate_(std::use_facet<std::collate<Ch> >(loc_))
    {}
    /// Construct a default sorter object, using the global locale.
    sorter() :
        loc_(std::locale()),
        collate_(std::use_facet<std::collate<Ch> >(loc_))
    {}
    /// Compare for less-than, for use as a comparison predicate with any
    /// standard algorithm.
    /// @param lhs left-hand side operand
    /// @param rhs right-hand side operand
    /// @return true if @p lhs < @p rhs in the given locale, false otherwise
    template<typename Tr>
    bool operator()(const std::basic_string<Ch,Tr>& lhs,
                    const std::basic_string<Ch,Tr>& rhs)
```

```
        {
            return collate_.compare(lhs.data(), lhs.data()+lhs.size(),
                                    rhs.data(), rhs.data()+rhs.size()) < 0;
        }
private:
    std::locale loc_;                    ///< cached locale
    const std::collate<Ch>& collate_;    ///< cached @c collate facet
};

/// Make a sorter object by deducing the template parameter.
/// @param name the locale name to pass to the sorter constructor
/// @return a new sorter object
template<typename Ch>
sorter<Ch> make_sorter(Ch const * name)
{
    return sorter<Ch>(name);
}

/// Main program.
int main(int argc, char* argv[])
try
{
    // Throw an exception if an unrecoverable input error occurs, e.g.,
    // disk failure.
    std::cin.exceptions(std::ios_base::badbit);

    // Part 1. Read the entire input into text. If the command line names a file,
    // read that file. Otherwise, read the standard input.
    std::vector<std::string> text; ///< Store the lines of text here
    if (argc < 2)
        read(std::cin, std::back_inserter(text));
    else
    {
        std::ifstream in(argv[1]);
        if (not in)
        {
            std::perror(argv[1]);            return EXIT_FAILURE;
        }
        read(in, std::back_inserter(text));
    }

    // Part 2. Sort the text. The second command line argument, if present,
    // names a locale, to control the sort order. Without a command line
    // argument, use the default locale (which is obtained from the OS).
    std::sort(text.begin(), text.end(), make_sorter(argc >= 3 ? argv[2] : "" ));
```

```
    // Part 3. Print the sorted text.
    std::copy(text.begin(), text.end(),
              std::ostream_iterator<std::string>(std::cout, "\n"));
}
catch (std::exception& ex)
{
    std::cerr << "Caught exception: " << ex.what() << '\n';
    std::cerr << "Terminating program.\n";
    std::exit(EXIT_FAILURE);
}
catch (...)
{
    std::cerr << "Caught unknown exception type.\nTerminating program.\n";
    std::exit(EXIT_FAILURE);
}
```

I caught you peeking. In spite of my warning, you tried to read the source code, didn't you? Just remember that I deliberately wrote this program in a complicated fashion to test your tools. By the time you finish this book, you will be able to read and understand this program. Even more important, you will be able to write it more simply and more cleanly. Before you can run, however, you must learn to walk. Once you are comfortable working with your tools, it's time to start learning C++. The next Exploration begins your journey with a reading lesson.