Intro to the C++ Standard Library

C++ has an enormous host of library functions, constants, and classes that can greatly simplify many programming tasks. While a substantial portion of the C++ Standard Library is beyond the scope of an introductory C++ class, much of the library is accessible to beginning or intermediate-level programmers. This handout serves both as an overview of the major divisions of the C++ library and as a reference list for those of you interested in exploring more advanced C++ topics.

The C Runtime Library

C++ was originally designed as a superset of the C programming language – that is, with very few exceptions, any code that you write in pure C should compile and run in C++. Consequently, for backwards compatibility, C++ absorbed C's runtime library. The C runtime library is composed of a set of header files whose name begins with the letter 'c.' For example, to access the C library functions that let you access and manipulate the date and time, use the header file `<ctime>`. Many (but by no means all) of the C runtime library has been superseded by other portions of the C++ library. For example, C's input/output routines like `printf` and `scanf` can be replaced with C++'s safer and more robust `cout` and `cin` streams. However, there are many useful C runtime library functions (my personal favorite is `tmpnam`) and I recommend that you take some time to look over what's there.

Because some of the C runtime library is unsafe when mixed with standard C++ or requires an understanding of the language beyond the scope of this class, we will not cover the C runtime library in great depth. However, if you're interested in learning to use the C runtime library, there are some great resources online, such as

- www.cppreference.com: A website covering both the C and C++ libraries. You might want to consider bookmarking this page as a quick reference because it's quite useful.
- www.cplusplus.com/reference/clibrary/: A categorized overview of the C runtime library that includes code examples for most of the functions and macros. As with the above link, this is an extremely useful website and you might want to consider bookmarking it. For those of you with high-end cell phones, consider adding it to speed-dial.

The IOStream Library

The IOStream library is C++'s way of reading and writing formatted input and output. IOStream includes functionality to read from and write to the console, files on disk, and even strings. In addition, it specifies a set of objects called *stream manipulators* that allows you to control the formatting and expression of data in a stream.

While our first lectures will cover a good deal of the IOStream library, we will not cover some topics such as binary file reading and random access, nor will we address some of the lower-level stream objects. For more information on these topics, you might want to refer to:

- www.cplusplus.com/reference/iostream/: cplusplus.com has a very good overview of the IOStream library that includes a handy class library and even offers a peek into the inner workings of the classes.

The String Library

For those of you with a background in pure C, the C++ string library might seem like nothing short of a miracle. The C++ `string` class is lightweight, fast, flexible, and powerful. In fact, it's so powerful and easy to use that it's one of the few standard C++ classes used in CS106B/X. For references for the string class, refer to the course reader and handouts in your CS106B/X class.

The Standard Template Library (STL)

The STL is a collection of classes that store data (containers), objects to access data (iterators), functions that operate on data (algorithms), and objects that manipulate functions (functors). An understanding of the STL is critical to fully appreciate how powerful and versatile the C++ language is. However, as is bound to happen with any powerful programming library, the STL is relatively confusing and requires a strong understanding of the core C++ language to fully use. Although we will spend several weeks exploring the STL, there simply isn't enough time to explore all of its facets and functionality.

If you're interested in exploring more topics in the STL, consider referring to these sources:

Scott Meyers. *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*. Addison-Wesley. ISBN 0-201-74962-9. This is widely recognized as one of the most useful books concerning the STL. Rather than serving as an introductory text on the subject, Meyers' book describes how to maximize efficiency and get the most for your money out of the STL.

www.cplusplus.com/reference/: A great (and free!) online reference covering much of the STL, including code samples.

Numeric Libraries

The numeric libraries, mostly defined in the `<numeric>` and `<valarray>` headers, are classes and functions designed for computational or mathematical programming. For those of you with a background in Python, this header includes classes that let you access arrays by using slices. We will not cover the numeric classes in CS106L, but for those of you who are interested, you may want to consider looking into:

msdn2.microsoft.com/en-us/library/fzkk3cy8(VS.80).aspx: Microsoft's `valarray` reference, which is one of the better coverages I've found.

Yang, Doagi. *C++ and Object-oriented Numeric Computing for Scientists and Engineers.* Springer. ISBN 0-387-98990-0. If you're interested in learning C++ in order to do computational programming, this is the book for you. It's a good introduction to C++ and includes many mathematical examples, such as solving ODEs. This book requires a mathematical background in linear algebra and calculus.

Memory Management Libraries

The C++ library also contains various objects and functions designed to help with memory allocation and deallocation. For example, the `auto_ptr` template class acts a “smart pointer” that automatically deallocates its memory, while the `set_new_handler` function can be used to set an emergency handler in case `operator new` can't find enough memory to satisfy a request.

While we may cover certain aspects of the memory management libraries, a complete treatment of memory management is an advanced topic far beyond the scope of an introductory class. If you're interested in some practical applications of the memory libraries, consider reading:

- www.gotw.ca/publications/using_auto_ptr_effectively.htm: The `auto_ptr` class is quite useful in simplifying your code and making it safer to use. However, it is not completely intuitive. This article is a great introduction to `auto_ptr` and offers several pointers and caveats.

Exception-Handling Libraries

C++ supports the notion of exception-handling by means of `try`/`throw`/`catch` blocks, as those of you familiar with Java might recognize. While we will cover exception handling later in the quarter, you may still want to explore the libraries in more depth than what is covered in this class. If you're interested in exception-handling in general, these resources might be useful:

- Sutter, Herb. *Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions.* Addison-Wesley. ISBN 0-201-61562-2. This book is an excellent resource on writing exception-safe code and is highly regarded in the C++ community. If you're interested in learning about writing code compatible with C++ exceptions, this is the book for you.
- www.boost.org/more/generic_exception_safety.html: While this site is primarily aimed at writing exception-safe container classes, it nonetheless provides an excellent introduction to C++ exception safety.

Locale Libraries

Many programs are designed for an international audience where notational conventions vary by location (for example, $1,234.56 in the United States might be written as 1.234,56 US elsewhere). The locale libraries offer a method for writing code that is location-independent. Locale functions are far beyond the scope of an introductory class, but those of you who are interested may consider the following source useful:

- www.cantrip.org/locale.html: This is one of the best introductions to locales that I've come across on the Internet. It uses a lot of advanced C++ syntax that might be confusing if you're not that familiar with the C++ language, so you might want to have a reference handy before proceeding.

Language Support Library

The C++ standard specifies the syntax and semantics of the C++ language in great detail, but it leaves many important decisions to individual implementers. For example, the size of an `int` or the behavior of a `double` holding too large a value vary on a system-by-system basis. To enable C++ programmers to determine the configuration of their systems, C++ provides the language support library, a set of classes and constants that contain information about the particulars of the current C++ implementation.

In CS106B/X and CS106L, you will not need to worry about the language support library. However, if you plan on working on a cross-platform C++ project where these sorts of details can matter, you may want to consider looking into:

- http://www.unc.edu/depts/case/pgi/pgC++_lib/stdlibcr/num_5679.htm: A complete reference for the `numeric_limits` class, which exports most of the information specific to a particular C++ implementation.

The Boost C++ Library

Although in CS106L we will concern ourselves only with standard C++, if you plan to pursue C++ beyond this class, you should strongly consider looking into the Boost C++ Library. Boost is one of the most prominent third-party C++ libraries, and several parts of Boost are being considered by the C++ Standards Committee for inclusion in the next release of C++. Once you've gotten the hang of the C++ Standard Library, you should strongly consider exploring Boost, especially since many parts of Boost seamlessly integrate with the C++ Standard Library. You can find the Boost C++ Library at www.boost.org.

Third-Party Libraries

For those of you with experience in modern languages like Java or Python, the C++ Standard Library might seem a bit limited. For example, C++ lacks a graphics and sound package, has no support for networking, and does not natively support windowing. To use features like these, you'll need to rely on third-party libraries, like Microsoft's Win32 API or the X Window System.

There are several reasons C++ opted out of a "monolithic library" strategy. First, since the C++ Standard Library focuses more heavily on data manipulation than presentation, C++ works on more platforms than other languages; you can use C++ to program both web browsers and microcontrollers. Second, some features like multimedia and windowing vary greatly from system to system. Standardizing these features would end up with a standard library catering to the lowest common denominator, something likely to please no one. Instead, C++ leaves libraries like these to third parties, so the resulting libraries are less portable but more powerful. Finally, libraries sometimes provide fundamentally different ways to approach programming in C++. Standardizing libraries of this sort would force C++ programmers comfortable with one programming style to uncomfortably adjust to another, and consequently libraries of this sort (for example, multithreading libraries) are left to third-parties.