Advanced Programming

Rensselaer CSCI-6090

December 6, 2001

This three-credit graduate course is being offered in the Fall 2001 semester. It meets 10-11:20 am Tuesdays and Fridays in LOW 3045 and on the Web at http://www.cs.rpi.edu/~musser/ap. The lectures and class discussions are an important part of the course but so is the Web site. Check it frequently for the latest information about the course, for lecture material, and for material not covered in the lectures or textbooks.

General Information

Instructor

Professor Dave Musser, Lally 313, 276-8660, musser@cs.rpi.edu
Office Hours: Wednesday 3:00 - 4:30 pm,
Friday 11:30 am - 1:00 pm, or by appointment

Teaching assistant

Arvind Venkatesan, Lally 3A (basement), venkaa@cs.rpi.edu
Office Hours: Wednesdays 2:00-3:00 pm or by appointment

Brief overview

Correctness, efficiency, and generality are three of the most important properties of successful computer programs. The main objective of this course is to develop intellectual tools that allow the programmer to attain a high degree of confidence that his or her programs have all three properties. We will focus on an approach called generic programming, whose techniques permit the definition of algorithms and data structures at an abstract level, in effect accomplishing many related programming tasks simultaneously. Methods of structuring a library of generic software components for maximum usability will be studied and practiced using the C++ language. An important tenet of generic programming is that generality should be achieved without losing efficiency, and we will consider performance testing and tuning techniques that aid in achieving highly efficient generic programs. Throughout the course we emphasize the importance of program correctness, with attention both to testing and debugging techniques and to methods of rigorous mathematical specification and proof.

Prerequisites

CSCI-2300 Data Structures and Algorithms

> *From the Rensselaer Catalog:* Data structures and algorithms, and the mathematical techniques necessary to design and analyze them. Basic data structures: lists, associative structures, trees. Mathematical techniques for designing algorithms and analyzing worst-case and expected-case algorithm efficiency. Advanced data structures: balanced trees, tries, heaps, priority queues, graphs. Searching, sorting. Algorithm design techniques: dynamic programming, greedy algorithms, divide-and-conquer, backtracking. Example graph, string, geometric, and numeric algorithms. Prerequisites: CSCI-1200 and MATH-1010. Fall and spring terms annually. *4 credit hours.*

The main background needed is programming experience and familiarity with data structures and algorithms, including basic efficiency analysis techniques. Programming will be done primarily in C++. The more advanced C++ language and library features will be thoroughly discussed and illustrated with example programs.

What it's not

This course is not on, but does draw ideas and goals from

- software engineering
- program synthesis
- analysis of algorithms
- programming language semantics
- object-oriented programming
- procedural programming
- literate programming (what's that, you ask?)

Homework, projects, exams, and grading

There will be homework assignments, a programming project, and one exam. Only a few of the homework assignments will be graded; they will count about 10% of the course grade.

For other homework assignments, you are not required to turn them in, but you are encouraged to do so, via email, at any time. You may also submit incomplete solutions and ask for help if you are stuck. These ungraded homework assignments are mainly are for you to get some practice with and test your comprehension of the ideas discussed in class. Looking at your solutions and questions also helps me assess which points need further discussion and which don't.

The programming project (which may be assigned in several parts) will count about 50% of the course grade. It will be a team project. Requirements on team size and further details will be given later. The final report and code for the project will be due near the end of the term.

The exam will be given approximately midterm and will count about 40% of the course grade.

Collaboration

You are encouraged to collaborate with other students in the class on the homework (**except homework 2**) and programming project. (For the project, you will form teams that work on the entire project together.) If you have worked on homework with other students and send me or Arvind email with solutions or questions, be sure to cc all students who have worked with you, so that we can easily cc our replies to all involved.

On the *exam*, collaboration is *not* permitted; all work must be your own.

I don't expect any incidents of academic dishonesty in this course, but any that do occur will be dealt with according the policies stated in the *Rensselaer Handbook*.

Topics

This schedule is subject to change—check the course Web site frequently for updates.

DATE	TOPIC	TEXTBOOK	READINGS	HOMEWORK / PROJECT	SUBMIT TO	WILL BE GRADED?
8/28	Objectives; Overview of Generic Programming					
8/31	Generic Programming & the STL	Accelerated C++	Chs. 0 & 1	All Ch. 1 exercises	Don't submit	No
9/4	Modern C++	Accelerated C++	Chs. 2 & 3	HW 1	musser@cs.rpi.edu by 9/6	No
9/7	More Modern C++; Word Frequency Program	Accelerated C++	Chs. 4 & 5	.	.	.
9/11	Introduction to BGL	BGL Book	Preface & Ch. 1	.	.	.
9/14	(Class cancelled)	.	.	.	.	.
9/18	GP and C++ Issues in BGL	BGL Book	Ch. 2	.	.	.
9/21	BGL Tutorial	BGL Book	Ch. 3	See 9/21 news	Don't submit	No

DATE	TOPIC	TEXTBOOK	READINGS	HOMEWORK / PROJECT	SUBMIT TO	WILL BE GRADED?
9/25	Writing Generic Algorithms; BGL Basic Graph Algorithms	Accelerated C++ Ch. 8; BGL Ch. 4	.	.	.	.
9/28	Algorithm Concepts for Standard Libraries	Handout	.	HW 2	venkaa@cs.rpi.edu by 10/12	Yes!
10/2	Performance Measurement	BGL Book	Ch. 11	.	.	.
10/5	Defining Abstract Data Types I; Project	Accelerated C++	Ch. 11	.	.	.
10/9	(No AP class; Mon classes)	.	.	.	.	.
10/12	Defining Abstract Data Types II	Accelerated C++	Ch. 12	.	.	.
10/16	Literate Programming; Concept Webs	.	.	.	.	.
10/19	Polymorphism and Virtual Functions	Accelerated C++	Ch. 13	**Project Proposal**	.	.
10/23	**Exam**	.	.	.	.	.
10/26	Memory Management	Accelerated C++	Ch. 14	.	.	.
10/30	Shortest Path & MST Algorithms	BGL Book	Chs. 5, 6	.	.	.
11/2	Connected Components & Max Flow	BGL Book	Chs. 7, 8	**Project preliminary report**	.	.
11/6	No class; project meetings	.	.	Project meetings	.	.
11/9	No class; project meetings	.	.	Project meetings	.	.
11/13	Correctness Testing	.	.	.	.	.

DATE	TOPIC	TEXTBOOK	READINGS	HOMEWORK / PROJECT	SUBMIT TO	WILL BE GRADED?
11/16	Performance Measurement	.	.	.	.	.
11/20	Tecton Formal Specification Language	Handout	.	.	.	.
11/23	(Thanksgiving)	.	.	.	.	.
11/27	Iterator & Container Concepts—Formal Descriptions	.	.	.	.	.
11/30	Brief reviews of project status (check email for schedule)	.	.	.	.	.
12/4	The Future of Programming	.	.	.	.	.
12/5	(Project deadline)	.	.	**Project final report**	.	.
12/7	Project help session	.	.	.	.	.
12/12	(Project deadline)	.	.	**Project final final report**	.	.

Resources

Textbooks

The following books are required; I will assign readings and exercises in them.

Andrew Koenig and Barbara E. Moo, *Accelerated C++*, Addison Wesley, 2000, ISBN 0-201-70353-X.

Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine, *The Boost Graph Library (BGL) User Guide and Reference Manual.*

A couple of comments on my choice of these books as textbooks for the course:

- If you are well versed in C++ you may think you don't need another C++ book. But you do! The Koenig and Moo book takes a completely different approach from other C++ books, stressing the use of the C++ Standard Library from the very beginning, which leads to quite different solutions to programming problems from approaches of other books (which tend to build upon knowledge of C and view the library as an afterthought).

- The second book isn't available yet from the publisher (Addison-Wesley) but I've obtained permission from the authors and publisher to make available photocopies of the manuscript for use in this course. These copies are available for purchase in the bookstore.

  Why a book about a graph library? BGL is an advanced application of the generic programming methodology that is the main focus of this course. The library itself is publicly available and is documented on the Boost organization's Web site.[1] The book contains this documentation but additionally contains many small examples of use of the library as well as good discussions of generic programming methodology. You'll need some basic knowledge of graph algorithms and data structures, but the concepts that go beyond those normally discussed in an elementary data structures and algorithms course are presented in the book (and it's easy to get practical experience with them by writing small programs that use the library components).

Although not required, the following books may be useful references:

Matt Austern, *Generic Programming and the STL*, Addison-Wesley, 1998. Much of the reference material in this book is also available on the Internet at the Silicon Graphics Standard Template Library Programmer's Guide Web site,[2] but the book has additional discussion of generic programming concepts.

David R. Musser, Gillmer J. Derge, and Atul Saini, *STL Tutorial and Reference Guide, Second Edition: C++ Programming with the Standard Template Library*,[3] Addison-Wesley, 2001. This edition of the book has been extensively updated from the original (1996) edition and has several new tutorial chapters.

Additional references:

Stanley B. Lippman and Josee Lajoie, *C++ Primer, 3rd Edition*, Addison-Wesley, Reading, MA, 1998, ISBN 0201824701.

Bjarne Stroustrup, *The C++ Programming Language, 3rd edition*, Addison-Wesley Pub Co, 1997, ISBN 0201889544.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms, Second Edition*, MIT Press, 2001, ISBN 0-07-013151-1.

Earlier editions of the two C++ books, or other pre-1997 books, are not useful due to the many changes in the language preceding its 1997 official standard. The first edition of *Introduction to Algorithms* (1990) remains a good reference, if you already have it.

[1] http://www.boost.org
[2] http://www.sgi.com/tech/stl
[3] http://www.awl.com/cseng/titles/0-201-37923-6

The C++ Standard

Here is a press release[4] that announced the final, official standard, ISO/IEC 14882-1998, "Information Technology — Programming Languages — C++," which is available for purchase[5] in either PDF (currently $18) or hardcopy form (don't ask).

The ANSI/ISO C++ Committee made some of the earlier drafts of the C++ Standard available to the public without charge. The last of these published drafts, dated December 1996, differs from the final official version in a few, mostly irrelevant, details. Thus you can use this draft to look up the definition of most language features and be reasonably confident of the answers (just don't bet the farm on it):

- hyperlinked version on the local file system
- PDF and Postscript versions[6] at AT&T Labs

[Note: don't attempt to print out the entire standard unless you have your own printer and paper—it's over 800 pages.]

Cygwin Tools (Including GNU C++)

The Cygwin Tools are a set of freeware compilers (mainly for GNU C/C++), other programming tools (such as debuggers), and utility programs (such as the BASH shell) for Windows 95/98/NT/2000. They provide many of the same useful tools you would find on a typical Unix platform.

If you have a Windows box and want to install the Cygwin Tools on it, download the installation file for your OS version from the Cygwin Tools[7] Web page.

C++ compilers

Information about C++ compilers particularly relevant to this course, especially compliance with the C++ standard, can be found at the following Web pages

- STL Tutorial Resources at Rensselaer[8]
- Boost Libraries Compiler Status[9]

GNU C/C++ compiler, g++ For this course, version 2.95.2 or 2.95.3 is recommended. Version 2.95.2 is installed on CS Lab Suns. Version 3.0.1 is also available, but may not be stable enough. The main standard-compliance problem with pre-3.0 versions is lack of namespace support. For the Windows platform, the Cygwin distribution currently includes g++

[4] http://www.ncits.org/standards/pr14882.htm
[5] http://webstore.ansi.org/ansidocstore/default.asp
[6] ftp://research.att.com/dist/c++std/WP/CD2
[7] http://sourceware.cygnus.com/cygwin/
[8] http://www.cs.rpi.edu/~musser/stl-book
[9] http://www.boost.org/status/compiler_status.html

2.95.3. This compiler is a just a command line compiler, but in combination with the compilation support commands of Emacs you can have a reasonable edit/compile/execute environment.

Borland compiler This compiler is available free for use on the Windows platform:

Borland C++ Builder Compiler[10]

It is a just a command line compiler (Borland offers a visual environment but not for free), but in combination with the compilation support commands of Emacs you can have a reasonable edit/compile/execute environment. Standard compliance in fairly good, including full support of namespaces. (If you use g++ (pre-3.0) as your primary compiler, you should also try your code with a compiler that fully supports namespaces to make sure you don't have namespace problems.)

C++ Web resources

- **C++ FAQs and Newsgroups**
  1. C++ Moderated Newsgroup: comp.lang.c++.moderated[11]
  2. Comp.lang.c++ Newsgroup FAQs[12]
  3. Bjarne Stroustrup's FAQs[13]
  4. Mathtools.net C++ FAQs[14]
  5. The comp.lang.c++ FAQs[15]
  6. General C++ FAQs[16]
  7. Landfield's C++ FAQs[17]
  8. Google Web-directory FAQs[18]
  9. Yahoo's C++ FAQs[19]
  10. C++ FAQs Lite[20] by Marshall P. Cline
  11. comp.std.c++ FAQs[21]
  12. More C++ FAQs[22]
- **STL**

[10] http://www.borland.com/bcppbuilder/freecompiler/
[11] news:comp.lang.c++.moderated
[12] http://www.landfield.com/faqs/by-newsgroup/comp/comp.lang.c++.html
[13] http://www.research.att.com/~bs/bs_faq.html
[14] http://www.mathtools.net/C++/FAQs/
[15] http://www.icce.rug.nl/docs/cppfaqs.html
[16] http://www.encrypted.net/~cpp/homepage/faqs.html
[17] http://www.landfield.com/faqs/C++-faq/
[18] http://directory.google.com/Top/Computers/Programming/Languages/C++/FAQs/
[19] http://in.dir.yahoo.com/Computers_and_Internet/Programming_Languages/C_and_C__/FAQs/
[20] http://cad.uccb.ns.ca/cpp_faq/
[21] http://reality.sgi.com/austern/std-c++/faq.html
[22] http://cplus.about.com/compute/cplus/msub9.htm

1. Silicon Graphics STL[23] (including a complete online STL reference manual)
2. STL Newbie Guide[24]
3. The Dinkum C++ Library Reference[25] (including a complete online STL reference manual, but also covering other important parts of the library such as the `string` class).

- **General C++ Web resources**

1. The home page of Bjarne Stroustrup,[26] the creator of C++.
2. The C++ Virtual Library[27]
3. Yahoo Index on C and C++[28]
4. Programming Language Resources[29]
5. C++ Users Journal Homepage[30]
6. Dr. Dobb's Website[31]
7. Microsoft Visual C++ Homepage[32]
8. C++ Annotations[33]
9. Common Questions about learning C++,[34] by Bjarne Stroustrup
10. Bjarne Stroustrup on C++[35]

Emacs

GNU Emacs, "the extensible, customizable, self-documenting real-time display editor," is available on most campus computers. If you want it on your own Windows box, visit the GNU Emacs[36] Web page at MIT.

If you are entirely new to Emacs or have never used it as a platform for compiling and running programs, you may find the following primer useful (for Unix platforms as well as Windows):

Program Development Using GNU C++ and Emacs: An Introduction,[37] by H. Allen Adams.

Emacs also has a built-in tutorial (type Esc-x help t) and "info system" (type Esc-x info).

---

[23] http://www.sgi.com/tech/stl
[24] http://www.xraylith.wisc.edu/~khan/software/stl/STL.newbie.html
[25] http://www.dinkumware.com/refcpp.html
[26] http://www.research.att.com/~bs
[27] http://www.desy.de/user/projects/C++.html
[28] http://www.yahoo.com/Computers_and_Internet/Programming_Languages/C_and_C_
[29] http://www.program.com/resources/proglang.html
[30] http://www.cuj.com/
[31] http://www.ddj.com/
[32] http://msdn.microsoft.com/visualc/default.asp
[33] http://www.icce.rug.nl/docs/cplusplus/cplusplus.html
[34] http://www.durhamc.on.ca/contact_us/yourth/bjarne.html
[35] http://www.cs.rit.edu/~mjl/oo_info/stroustrup_on_Cplusplus.html
[36] http://www.gnu.ai.mit.edu/software/emacs/emacs.html
[37] emacs.txt

LaTeX

LaTeX is available on CSLab, RCS, and probably most other campus computers. To apply it to latex source file `mydoc.tex`, run

```
latex mydoc
```

This produces a DVI file, `mydoc.dvi`, which can be converted to a Postscript file, `mydoc.ps`, with

```
dvips -o mydoc.ps mydoc
```

or to a PDF file, `mydoc.pdf`, with

```
dvipdfm mydoc
```

Pdflatex is a tool that produces a PDF file directly from a LaTeX source file `mydoc.tex`:

```
pdflatex mydoc
```

produces `mydoc.pdf`. Using a LaTeX package called `hyperref`, e.g., with

```
\usepackage[plainpages=false,pdftex,colorlinks,backref]{hyperref}
```

causes cross-references and citations to appear as hyperlinks in the resulting PDF file.

To download a LaTeX system (free) for use on a Windows box, visit

> MiKTeX[38]

The MikTeX distribution includes utilities `dvips`, `dvipdfm`, and `pdflatex`.

The original LaTeX manual, by the system's creator, is

> Leslie Lamport, *LaTeX: A Document Preparation System User's Guide & Reference Manual*, Addison-Wesley, 1986, ISBN 0-201-15790.

Another possibility is *First Steps in LaTeX*, by George Gratzer.

Beginning LaTeX users (and some veterans) sometimes make errors that cause no complaint from the LaTeX processor but do result in poor formatting. The following short note discusses some common problems and how to avoid them:

> D. R. Musser, Elements of LaTeX Style (Postscript file), November 24, 1998.

[38]http://www.miktex.org/

Boost Libraries

The Boost Organization's[39] C++ Libraries are installed on the CS file server, in /cs/musser/public.html/ap/boost_1_24_0. They consist entirely of source code and documentation, with the exception of one subdirectory as discussed below. In most cases you can compile programs that include headers from the Boost libraries, such as most programs in the BGL textbook, simply by including the following option on the compilation command line:

```
-I/cs/musser/public.html/ap/boost_1_24_0
```

If a program calls the BGL `read_graphviz` function, for input of graph data structures in graphviz form, another command line parameter is needed to link to an additional library. [9/17/2001: This part of the library is currently not fully set up on the CS file system but should be soon. For now, it should not matter as long as your program doesn't use `read_graphviz`.]

For installing the Boost libraries on a Windows box, there is again no problem as long as you avoid `read_graphviz`. If you want to use it, you must compile the code in the subdirectory boost_1_24_0/libs/graph/src. I was able to compile it with the Makefile in that subdirectory, but only after updating from g++ version 2.95.2 to 2.95.3 (which I did by updating my entire Cygwin Tools installation to version 1.3.3).

See also pp. xx and xxi of the BGL textbook preface.

Accelerated C++ Source Files

Source files and related material for the *Accelerated C++* textbook are available at the site (given on p. xiv) http://www.acceleratedcpp.com. Here are the files as downloaded from the link

> Source code (packaged for Unix and similar systems, tar file compressed with gzip).

These files have restricted access: in the pop-up access dialog box enter the username `fall2001` and the password mentioned in class (or send email to the instructor or TA to get it).

- Accelerated C++ Source Files

Nuweb and Pdfnuweb

Nuweb is a relatively simple but powerful tool that supports the literate programming style of simultaneous coding and documenting. It was originally developed by P. Briggs and has been extended and modified by others, including J. Ramsdell, M. Mengel, and D. Musser. Pdfnuweb is an extension of nuweb, developed by R. Loos, that adds hyperlinking to the code part definitions and uses in the documents it produces, in the form recognized by the hyperref package for pdflatex. In the information that follows, everywhere nuweb is mentioned the corresponding statement for pdfnuweb is also true, except as noted.

[39] http://www.boost.org

On CSLab *SUN* workstations run nuweb or pdfnuweb using the command

`nuweb file`

or

`pdfnuweb file`

where `file.w` is a nuweb source file. [**Added 11/16/2001:** The version of pdfnuweb currently installed on CSLab SUN workstations has a bug in the way it handles file names with underscore characters. Until a new pdfnuweb version is installed, avoid using underscores in file names, or use nuweb instead.] [**Added 11/18/2001:** It has been installed.]

On RCS *SUN* systems, run nuweb with

`/home/96/mussed/public/nuweb file`

This is only for RCS SUN workstations; there's currently no version compiled for IBM workstations, and no version of pdfnuweb on either SUN or IBM workstations. Nor is pdflatex installed on RCS SUN or IBM workstations (but latex is).

For running nuweb or pdfnuweb on a Windows PC, download a binary executable, nuweb.exe or pdfnuweb.exe. [**Added 11/16/2001:** If you downloaded pdfnuweb.exe before 11/16/2001, download it again. A bug has been corrected in the way pdfnuweb handles file names with underscore characters.]

[**Added 11/18/2001:** For running pdfnuweb on an Linux box with an Intel Pentium processor, download a binary executable, pdfnuweb. Thanks to Noboru Obata for compiling this binary.]

The following document describes the nuweb command set (which is very simple) and is also useful as a moderate-size example of the literate programming style: it documents the design and implementation of nuweb itself.

> Nuweb Manual, a PDF file, about 80 pages, also available as a gzip-compressed Postscript file).

You probably want to print out just the first section of the manual, which describes the commands. (E.g., view it with Adobe Reader or Ghostview and select the pages you want for printing.) This version of the nuweb manual still only describes nuweb, not pdfnuweb.

There is a bug in this version of nuweb that makes it necessary to quote `@` characters by prefixing an `@`, writing `@@`, *within nuweb parts* as well as outside of parts. Likewise, if you want a `$` to appear in a part, you must prefix it with `@`.

A nuweb mode for Emacs is provided by nuweb.el (an elisp file that can be loaded into Emacs with `EscX load-file`).

Homework

Homework 1

For this exercise set, send your answers to me in an email message no later than 10pm Thursday 9/6. If you only have partial answers, or answers you are not sure of, send them in anyway and point out any places where you had problems. I won't assign grades to these but I will try to look them over and comment by email or in class on anything you have questions about or difficulty in answering.

You may want to work in groups on these exercises. This is fine as long as you don't depend too much on other members of the group—you may encounter similar questions when you take the exam, and you'll be on your own then. Include the names of all members of the group in any email you send me, and cc the message to all members; I'll do the same with my replies.

Do Exercises 2.10, 3.1, 3.3 from *Accelerated C++*.

In case you don't have this book yet, the text of these exercises is reproduced below. However, *you should obtain the book as soon as possible so that you can do the reading assignments in it.* What is an acceptable answer to an exercise from the book depends in part on the context provided by the discussion in the book. For example, a solution to Exercise 3.3 written in C, or even in C++ without making good use of the C++ Standard Library, would *not* be an acceptable solution. The book places strong emphasis on making effective use of the library as well as the non-C features of C++, and any programs you write in response to the book's exercises (or any other assignments in this course) should reflect that emphasis, even if the assignment doesn't explicitly call for it. In future homework assignments it will be assumed that you have the book and the text of exercises taken from it will not be reproduced here.

2-10. Explain each of the uses of `std::` in the following program:

```
int main()
{
    int k = 0;
    while (k != n) {      // invariant: we have written k asterisks so far
        using std::cout;
        ++k;
    }
    std::cout << std::endl; //std:: is required here
}
```

3-1. Suppose we wish to find the median of a collection of values. Assume that we have read some of the values so far, and that we have no idea how many values remain to be read. Prove that we cannot afford to discard any of the values that we have read. *Hint:* One proof strategy is to assume that we can discard a value, and then find values for the unread—and therefore unknown—part of our collection that would cause the median to be the value that we discarded.

3-3. Write a program to count how many times each distinct word appears in its input.

Homework 2

This homework assignment is **due on Friday, October 12 at 11:59:59 pm, and it will be graded.** (Counts about 10% of the course grade.) **On this homework, no collaboration is allowed; all work must be your own.**

Late submissions: Assignments turned in up to 24 hours after the due date will receive a 10% penalty. After that, late assignments will receive a 20% penalty up until the time solutions are posted or discussed in class, after which they cannot be accepted at all. A solution to this assignment will be posted at 6pm on Thursday, October 18. Students should not discuss their solutions or attempts at a solution with any other student before that time, even if they have already submitted their work, and even if they have already received a grade on it.

See also the **expanded submission instructions**, **additional notes**, and **frequently asked questions** at the end of this section.

Topological sorting A topological sort of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before v in the ordering. Note that a given graph generally has more than one topological ordering.

Chapter 3 in the BGL tutorial describes a generic topological sort algorithm `topological_sort` that is based on supplying an appropriate visitor to depth first search. The documentation of this algorithm is given in section 13.2.5 of the text. However, for this assignment we shall be using a different approach.

Another way to perform topological sorting on a directed acyclic graph $G = (V, E)$ is to repeatedly find a vertex of indegree 0, output it, and remove it and all of its outgoing edges from the graph. This can be implemented as an $O(|V| + |E|)$ algorithm (same as the BGL algorithm in section 13.2.5), by keeping proper track of vertices with indegree 0; i.e, at any point it is needed you must be able to find such a vertex in constant time.

Algorithm and testing method development There are three principal functions to be developed:

1. Implement the above approach (in C++ using BGL), as a generic algorithm with basically the same interface as the BGL algorithm as described in section 13.2.5. Note that that interface does not assume bidirectionality of the graph representation, so your algorithm will have to initially compute and keep track of indegrees. Your algorithm should have an $O(|V| + |E|)$ time bound.

   Note: There is an error in the specification in Section 13.2.5, in saying that the input can be “a directed or undirected graph.” It must be directed.

2. Specify and implement an topological sort acceptance testing function. This is a boolean valued function that accepts an acyclic directed graph and a linear sequence of vertices as inputs, and returns true if and only

if the sequence is a topological ordering of the graph. (This function will be needed in order to check the result of running your topological sort algorithm on large graphs, as required in the second part of the assignment, since for these graphs it will be too difficult to check the result by hand.)

Note that you *cannot* implement this function by calling another (trusted) topological sorting algorithm and comparing its result with the given ordering, since the other algorithm may produce a correct but different ordering.

In general, an acceptance testing function should be written in as simple a way as possible, so that its own correctness is easily verified by inspection of the code. But if it is to be applied to large data sets you do have to be careful that it doesn't take more time than the algorithm it is being used to check! (Thus in this case, it should have an $O(|V|+|E|)$ bound, whereas an "obvious" method might take $O(|V|^2+|E|)$.)

3. Specify and implement a function that generates random acyclic graphs of a given size and denseness. This function will be needed in order to test your topological sort algorithm on large graphs. You should be able to control the denseness of graph by varying the probability of edge creation in the graph. For example a probability of 0.7 would mean that, out of all the 10 possible edges in the graph, 7 of the edges are actually created, on average.

   One problem in creating random graphs is that cycles may appear in the graph. To make sure you create random *acyclic* graphs, use the following method:

   (a) Let the vertices be numbered from 1 to n.

   (b) Whenever adding an edge to vertex k, always choose one of the vertices numbered between $k+1$ to n. The choice of vertices in this range is done randomly. This will ensure that there is no edge going back to any of the parent vertices in the graph.

   Although this method eliminates some randomness in the graph creation, it still is a reasonable way for purposes of algorithm testing to create large acyclic directed graphs.

Testing methodology Test your programs as follows:

1. Start testing your algorithm with small graphs for which the result can be hand-checked. Chapter 3 (File Dependency graph) and section 13.2.5 each have such an example.

2. Once you get the above part running you can go ahead and test your algorithm on large random graphs. In such tests you will use your random graph generator to generate a graph, apply your algorithm to it, and check the correctness of the result with your acceptance

testing function. Among the tests you do, include one with a graph with 1,000 vertices and about 5,000 edges (edge probability **0.01**). (Originally the assignment said 0.05, but as as noted in class on 10/5, this was incorrect. A complete DAG with N vertices has $N(N-1)/2$ edges, so for N =1,000, about 500,000 edges.)

Submission Submit all of your code including the program or programs that set up the test cases. Add a README file that briefly describes the programs. It should also describe the results of the testing, explicitly mentioning any case in which one of your programs does not work as specified or does not finish or takes an excessive amount of time. Put all the files in a directory and zip the directory. The directory should be named the same as your rcs-id. E.g., my rcs-id is `mussed` (different from my CS lab id), so I would put my files in a directory named `mussed` and execute (from the directory level just above it)

```
zip -r mussed mussed
```

This would create a zip file named `mussed.zip`. Mail the zip file to Arvind (venkaa@cs.rpi.edu) on or before the due date. (Note that zip is available on campus Unix systems as well as Windows boxes.)

See also the **additional submission requirements** posted by Arvind on his Web homepage.[40] (Check them again for any updates just before you submit.)

Additional Notes

1. Data files for one of the suggested small example graphs are available in the 9/21/2001 news article. As I mentioned in class, the graph created by reading in these files is slightly different from the graph in Figure 3.1 in the BGL book. The reason it is different is just that when I assigned numbers to the vertices in the file, I used the numbers 1 through 15, whereas I should have used 0 through 14 to make them convenient for use as vector indices. So I added a new vertex 0 and gave it the name foo.cpp. This choice of name was a slight mistake too, since there was already a name foo.cpp in the data. This causes no harm in terms of the way the algorithm works (it is working on the distinct vertex numbers 0 through 15), but makes the output look a little strange. I have since changed the name of vertex 0 in the online file.

2. For certain operations a hash table is the data structure of choice, since it offers (average case) constant time insertion and lookup. For example, checking equality between two sets of values of an arbitrary type (not just integer values) can be done in linear time. If you want to use a hash table for such purposes, use the hash_set or hash_map containers in the GNU C++ library (which are adapted from SGI

[40] http://www.cs.rpi.edu/~venkaa

STL). Unfortunately the C++ standard does not require such hash tables to be included in the standard library, so your code that uses them may not be portable. If you wish to use another compiler or maintain portability to other compilers, you may use the standard set or map container instead. This will likely introduce an extra log factor in the computing times of your algorithm; e.g., your acceptance testing function could become an $O(|V| \log |V| + |E|)$ algorithm instead of meeting the stated requirement, $O(|V| + |E|)$. This will be permitted without penalty. You should *not* program your own hash table.

Questions Asked (Some of Them Frequently)

1. Q.

```
I'm working on the homework and I've run into a snag

if I define my graph to be as follows:

typedef adjacency_list<listS, vecS, directedS> Graph
Graph G

then call:

in_degree(*vi,G)

where vi is a vertex_iterator I get a compilation error stating that
Graph doesn't have a member m_in_edges.  I read further in the BGL
book in section 12.1.4 and it says that only BidirectionalGraph
supports in_degree which conflicts with what is written in the
assignment where it says the graph needs to be directed.
```

A. Remember, the assignment says "Note that that interface does not assume bidirectionality of the graph representation, so your algorithm will have to initially compute and keep track of indegrees." If the graph were bidirectional you could get indegrees almost for free just from the iterators for incoming edges. Instead you'll have to compute them initially by traversing the graph and counting how many edges have a vertex as its target. Then you need to maintain the indegrees as the algorithm proceeds.

2. Q.

```
I have question about homework 2.  I've just implemented my idea
and tried to run against the graph on page 206 of the BGL book.
But my program couldn't process the graph correctly.

The graph seems to have a cycle.  A cycle detection function
introduced in page 54 of the book also says the graph is cyclic.
```

```
Should my program process it correctly?
```

A. Get rid of the (f,f) edge. (Apparently the topological sort algorithm in the BGL works even if you have such a self-edge, but the new algorithm you implement for this assignment doesn't have to.)

3. Q. `I nearly finished my source code, but couldn't figure out how to compile it on VC++ or is VC++ ok for this program?`

   A. As I've said in class, you might be able to compile BGL programs with VC++ — or you might not, since VC++ doesn't support the C++ standard very well. If you don't want to struggle with whatever problems come up, you should use g++. You can do this on the CSLab Sun workstations or on your Windows box if you download and install the Cygnus tools (which you really should have anyway since otherwise a Windows box is only marginally useful for CS purposes). See the course Web page articles on compiling programs with the boost lib using g++.

4. Q.

```
For checking if the topological order that i generate is correct or not,
is the following a sufficient condition to check...
 1. For all the vertices, store their indegrees in a vector...O(V)
 2. For each vertex in the topologial order generated, check if the vertex
has indegree 0 and for all the outedges decrement the indegrees of the
target vertex in the vector generated in step 1. ...O(V+E).
 For the ordering to be correct, the vertex being checked should have
value 0 in the indegree vector.
Am i missing any other condition that i should check ?
```

   A. Sounds like you are trying to do the check by programming the same algorithm. This won't work; if you have a mistake in your original algorithm you likely have the same mistake in the version you use for checking. To write the checking function, you should concentrate on testing what the definition of a topological ordering says: given a DAG G and a linear sequence S, check that for every (u, v) that is an edge in G, u appears before v in S. Also check that all vertices in G, and only those vertices, appear in S.

5. Q. `Since we are writing a generic random graph generator and the node of the graph can be any type (therefore generic), how can our program know how to detect any difference between those nodes?`

   A. It isn't required that your random DAG generator be a generic algorithm. It's okay if it only produces graphs with integers 0, 1, 2, $\ldots$, $N-1$ as vertex descriptors.

6. Q. `If our DAG generator isn't generic and we produce graphs with integers as vertex descriptors, doesn't that mean that our sort function only needs to accept int type? Why even bother with generic topo sort?`

   `Can we write our sort function given the knowledge that the vertex descriptors are integers?`

   `Because I don't see how we gonna write our sort function to be generic without using a std::pair for the vertex.`

   A. No, the purpose of testing with large random graphs is to mainly to make sure that your algorithm is reasonably efficient; it is not intended to test for other things such as that your algorithm is generic. The requirement that the algorithm is generic still holds. Thus you should *not* assume that the graph's vertex descriptors are integers. Nor should you assume that they are std::pairs. If in your algorithm you need to use vertex descriptors as unique ids, try using a hash_map or map with the vertex descriptor type as the key type.

7. Q. `For homework 2, when designing the randomly generated graphs, should they be fully connected? I thought we should do so, then decided maybe it's not needed. Which should we do? Full connections, or allow multiple chunks of graphs?`

   A. Fully-connected graphs are not required.

8. Q. `I have tried to get the program in section 13.2.5 to work, and just can't get it to compile. I know that it has to do with the actual call to topological_sort that I am making, but even after looking at the boost code that you have shared, I can't figure out what is wrong. I'm sure that I'm totally missing something important, but if I can't even get the example going I have no hope for the homework. I have included the source code that I am using.`

   A. It may be that your problem is just that you need to increase template instantiation depth. I tried compiling your program with -ftemplate-depth-30 on the command line, and it compiled and ran properly. [I shortly afterward got a reply that said it did solve the problem.]

9. Q. `In part 1, do we need error checking that tests whether the graph is acyclic, or is it just assumed that only acyclic graphs will be passed in?`

   A. You can assume that acyclic graphs will be passed in.

10. Q. `I'm finishing up the random graph generator (which is actually pretty easy), and I'm pretty sure that the formula given on the website for maximum number of edges on a DAG with N vertices should be N(N - 1)/2, not N(N + 1)/2. With the latter formula, self-edges are included. A quick example is a graph with 1`

vertex must have 0 edges, but by the formula, 1(1 + 1)/2 = 1. Is this correct?

A. Yes, it should be $N(N-1)/2$ for the maximum number of edges on a DAG. This has been corrected in the online problem statement.

11. Q. Also, does the topological sorting function have to work if listS is selected for the vertex list, i.e., `adjacency_list<vecS, listS, directedS>`? It seems that my algorithm would be quicker with vecS for the vertex list, because I am not doing any insertions or removals. Although I'm not sure if it'd still be considered generic if it didn't accept listS. What do you think?

    A. The topological sort should work even if you have `listS` for the vertex list. How do you know? Because the specification of the BGL topological sort in Section 13.2.5 says that "the graph type must be a model of `VertexListGraph` and `IncidenceGraph`." If you look at the requirements of these concepts (see p. 159), you see that `listS` for the vertex list satisfies those requirements.

12. Q. I just realized that my sorting function gives the topological sort in its normal order (not reverse). Should I reverse it inside the sorting function to keep it exactly the same as the topological sort from the BGL book?

    A. Either way will be accepted without penalty.

A Solution to Homework 2

These files have restricted access: in the pop-up access dialog box enter the username `fall2001` and the password mentioned in class (or send email to the instructor or TA to get it).

- Individual files:

  Makefile

  makefile-dependencies.dat

  makefile-target-names.dat

  random_acyclic_graph.hpp

  test_random.cpp

  test_random_acyclic_graph.cpp

  topo_sort6.cpp

  topological_sort.hpp

- Zip file for downloading all files:

  toposort.zip

Exam

The exam was given in class on Tuesday, October 23. These files have restricted access: in the pop-up access dialog box enter the username fall2001 and the password mentioned in class (or send email to the instructor or TA to get it).

- Practice Exam

  Note: there was an error in the Practice Exam in question 13:

  vector<vertex_t> should be vector<degree_size_t>.

  Make this correction in your hard copy. It has been corrected in the on-line version above (and it was already correct in the answer key below).

- Practice Exam—Some Answers
- Exam
- Exam—Some Answers

Project

The main purpose of the project is furtherance of the overall goal of the course: to develop intellectual tools that allow the programmer to attain a high degree of confidence that his or her programs have the important properties of generality, correctness, and efficiency. The projects are a way of consolidating and putting into practice some of the principles covered in the course, including not only programming techniques and language knowledge but also understanding of important computer science concepts and how they are related.

Project requirements

- Develop one or more generic software components, designing them to fit into the STL or BGL framework of containers, generic algorithms, iterators, function objects, adaptors, allocators, and visitors. "Develop" includes doing concept development,[41] design, implementation, experimentation, testing, and documentation.
- The emphasis of your project may be in any one of the following three areas:
  1. Developing new components, with functionality distinct from any components already in STL or BGL.
  2. Developing alternative implementations for existing STL or BGL components, based on different algorithms or data structures with different performance profiles (improving on existing implementations for a definable, nontrivial class of inputs).
  3. Developing software tools that support generic programming activities such as compiling (e.g., help with error messages), debugging (e.g., help with interpreting run-time execution states), or general understanding of generic algorithms and data structures (e.g., documentation aids, algorithm animations, etc).
- Observe the documentation requirements and guidelines given in a later section.
- For a generic algorithm, be sure you are starting with an efficient concrete algorithm; there is no point in making a generic bubble sort. The same is true for other kinds of components.
- Testing should be extensive, including checking correctness of small cases, including boundary cases such as empty sequences, and measuring performance of large randomly generated inputs of a variety of sizes (possibly up to 100,000 or a million elements, depending on the algorithm).

[41] "Concept" is used here with the technical meaning of a "family of abstractions that satisfy a common set of requirements," as exemplified in the Silicon Graphics Standard Template Library Programmer's Guide and formally developed in the Tecton concept description language.

- Correctness for large inputs should also be checked, using an acceptance routine (e.g., to check a sort routine, use an acceptance routine that checks to see if the result is in order, or better, checks to see if it is exactly the same output as that from another sort routine known to be correct).
- Timings of fast algorithms on moderate size inputs (a few thousand elements) may need to be repeated for some number of trials and averaged, for accuracy.
- D. R. Musser, Measuring Computing Times of Generic Algorithms.[42] contains advice and software tools for accurate performance measurement of generic algorithms. Alternative (and generally better) methods will be a topic of discussing in an upcoming lecture.
- For further ideas and discussion, see the test plan and results subsection of the Documentation Requirements and Guidelines section.

- Experiment with different potential optimizations of your generic component(s). For each component, attempt to find, based on performance measurements, the best version, under the constraint that it should be generic. Include in the Design section of your documentation discussion of why the selected and documented version is better than alternatives. For algorithms, also compare performance with other algorithms; if your algorithm isn't the best for all kinds of inputs, try to describe the class of inputs for which it is best, in the Complexity section.

- You must form **teams of three, four, or five** people and submit one implementation and report for the team. You may divide up responsibilities for the project as you wish, but each member of the team should keep fully informed about the design decisions, experimental results, testing results, and documentation.

Important dates

- By **Friday, October 19**, inform me by **email** of what you plan to do for your project (one message per team). Also list your sources for concrete algorithms, data structures, or other information you plan to use in the project. (This list doesn't have to be final.)

- By **Friday, November 2**, submit by email a preliminary report on your project, as a basis for discussion in individual sessions with me the following week. Guidelines for the contents of this report are given in the next section.

- During the week of **November 5**, I will hold 40 minute sessions in my office to meet with each team about the current state of their project.

[42] http://www.cs.rpi.edu/~musser/gp/timing.html

Four of these meetings will be during the regular class periods; we will not meet as a class in this week.

The meeting schedule is given in a later section.

- On **Friday, November 30**, in class, submit your "final" document. ("Final" is in quotes because I may ask you to revise it and resubmit it.) Also submit it by email. [**Added 11/27/2001:** See revised date and submission instructions.]

Preliminary report contents

What to submit in your November 2 preliminary report:

- Brief overview description of the components you are developing, and the key idea(s) of the generic algorithm(s) or data structures involved (about one half to one page total), including why you think these components might be useful.

- Brief description of your test plan—what kind of tests you will set up to test for correctness and performance (about one half to one page total).

Projects, Teams, and Meeting Schedule

Teams and projects

The project titles are tentative, based on what has been proposed by email or in limited discussions. They may be revised after receiving your preliminary reports.

1. Greg Buhler, Matthew Goodyear, Alexander Horwatt, Noboru Obata, Eunyoung Seol, *A splay tree implementation of the std::map container*

2. Anand Balan, Ruchi Dayal, Omesh Tickoo, *A generic algorithm for finding long paths in a graph*

3. Yebin Luo, Rui Mu, Jiahe Si, Hua Yang, Ki Yim, *A generic Floyd-Warshall algorithm for the all-pairs shortest paths problem*

4. Niharika Mateti, Chandrika Palagiri, Saroj Yadav, *A generic path problem solver for directed graphs*

5. Bill Katt, JJ Johns, Mike LaSpina, Matt Schumaker, Jeremy Shute, *Generic networked IO*

6. Jing Han, Kewei Huang, Yu Shao, Jianliang Yi, *Maximum cardinality matching in general graphs*

7. Vikram Agrawal, Jitendra Deshpande, Yogesh Girdhar, Adnan Saifee, *Generic planar graph algorithms*

8. Mital Chandihok, Scott Hunter, Kevin Ring, *Heterogeneous containers*

Meeting schedule for project discussions

Tuesday, November 6	10:00 - 10:40am	**Team 6**
.	10:40 - 11:20am	**Team 3**
Wednesday, November 7	10:40 - 11:20am	**Team 2**
Thursday, November 8	10:00 - 10:40am	**Team 8**
.	10:40 - 11:20am	**Team 7**
.	2:00 - 2:40pm	**Team 4**
Friday, November 9	10:00 - 10:40am	**Team 5**
.	10:40 - 11:20am	**Team 1**

These meetings will be held in my office, **Lally 313**, or in **Lally 311**. If both doors are closed, knock on my office door first.

All members of the team must attend the meeting.

Documentation Requirements and Guidelines

How should generic software be documented? This is a discussion of the problem in general, plus some specific requirements for the project reports for this course. We start with a general discussion that mostly applies to nongeneric as well as generic software components.

User documentation vs. design documentation vs. code documentation

- **User documentation**: *What* a user needs to know to use the software components to build applications; concerned with interface and abstract semantics, *not* with implementation. User documentation is best separated into a User's Guide, which introduces the software gently through key ideas, examples, and other tutorial material, and a Reference Manual, which more precisely and completely fills in the details.

- **Design documentation**: *Why* the software is designed as it is, mainly for benefit of present and future maintainers of the software; giving the rationale for interface and abstract design decisions, for the choice of particular algorithms and data structures, versus alternatives; this documentation is not for the user.

- **Code documentation**: *How* the components are implemented—coding details and design decisions that are not obvious from the code; this is not for the user but mainly for later maintainers of the code.

User documentation should not refer to the implementation design, except perhaps to help explain how components fit together (the framework); if it refers to the algorithms, data structures, or actual code, it should be only for the benefit of users who may like to know, but the information given should not be essential to proper use of a component.

For example, consider the SGI STL sort[43] function description, which describes

- what it does
- performance bounds
- stability properties

Whether it uses quicksort, merge sort, radix sort, heapsort, etc., is not (necessarily) of interest to the user. But note that for this to be true, all three of the above parts of the description must be present.

It might be mentioned in the documentation that (a version of) quicksort or introsort is the algorithm used to implement the component, but it should not be necessary for the user to know this.

User documentation is also a contract that must be maintained when code (and perhaps the whole implementation design) later changes.

User documentation of generic software components

The challenge is how to describe the abstract semantics in its full generality: just how widely is the component adaptable? This often requires creativity, and several tries—you might not realize how general the component is until you attempt to describe it carefully. Entirely new ways to use it may be suggested.

Also you may have to define new concepts, by means of sets of requirements, in order to organize your descriptions coherently.

The remaining discussion in this section will emphasize techniques for describing a component in its full generality and avoiding descriptions that would unnecessarily exclude certain uses.

Once again, we look at examples of user-level generic component documentation from the SGI Web pages on STL.[44]

There are a few places where the SGI documentation does not describe components in their full generality, such as the `merge` function (see below).

Specifying concepts The following definitions briefly summarize the terminology used in documenting generic software using concepts.

- Concept description: set of type requirements
- Concept: set of types that meet the described requirements
- Model of a concept: any type that satisfies the requirements
- Refinement of a concept: another concept with the same requirements plus additional ones (similar to class inheritance). If concept B is a refinement of concept A, then B has fewer models than A.
- Concept hierarchy (collection of concepts and their refinement relation)

[43] http://www.sgi.com/tech/stl/sort.html
[44] http://www.sgi.com/tech/stl/

Specifying functions Consider the STL generic merge[45] function. Do you see how it can it be used more generally than the SGI description implies?

An effective way to describe the input-output relation computed by a function is in terms of preconditions, postconditions, and effects.

precondition: a predicate that states what condition (relation of input variables) must be satisfied by any call of the function; sometimes called a *requires* clause.

postcondition: a predicate that states what condition (relation of input and output variables) that is satisfied after the function call is completed, provided the precondition is satisfied; sometimes called an *ensures* clause.

effects: a list of changes the function makes on variables other than those considered to be outputs of the function; sometimes called a *side-effects* or *modifies* clause.

A function may throw an exception, in which it does not return in a normal manner and its input-output relation as described in the above terms does not apply. Any condition under which a function throws an exception is in general unrelated to the function's precondition:

- If the precondition is violated, an exception is not necessarily thrown. [Give an example.]
- Even if the precondition is satisfied, an exception may be thrown. [Give an example.]

When it may be necessary to explain design decisions Sometimes, if some aspect of a component interface design is different from what is usually described in textbooks or seen in other libraries, it is helpful to the user to explain it. For example, the SGI documentation of the STL queue[46] adaptor takes the trouble to explain why the `pop` member function has a void return type.

Tips Finally, you can help the user by including *tips* for use of components. Tips go beyond how to use a component to describe how to use it well. Example: in the SGI description of the STL vector[47] component, see the tips relating to storage management given in the notes at the end of the page.

Contents and organization of project final reports

Software documentation is often published in separate volumes called User's Guide, Reference Manual, and Design Document. Results of testing software

[45] http://www.sgi.com/tech/stl/merge.html
[46] http://www.sgi.com/tech/stl/queue.html
[47] http://www.sgi.com/tech/stl/Vector.html

are not usually published outside of an organization's development group, and (unfortunately for later maintainers) may not be documented at all. For the projects for this course, prepare a single document with the following sections:

- User's guide
- Reference manual
- Design issues
- Source code
- Test plan and results

The user's guide contains the user level documentation and may contain tutorial material (such as extended discussion of examples), but for purposes of this course, tutorial material is optional.

The complete source code of a software product is not usually published, but in this case your project reports should include it. It is *required* that the code be presented in the literate programming style, using the Nuweb tool. The code that you compile and test must be the code that is extracted from the Nuweb source file for your project. You should submit your report containing the code and the other sections in both hardcopy form (resulting from running Nuweb source file → nuweb → latex → dvips → laser printer, or, better, Nuweb source file → Pdfnuweb → pdflatex → laser printer) and as an email message containing only the single Nuweb source file.

The SGI web pages contain mainly overview and reference material, and some discussion of design issues; they contain only minimal tutorial material and lack any discussion of testing. They point to the code but do not use literate programming in presenting the coding details.

User's guide The User's guide should begin with an overview section. The overview section is a crucial part of the documentation and you should make sure it really does give an overview. Issues in writing a good overview include:

- Who is the intended audience.
- What is the right level of discussion for that audience, *in the overview* (you can change to a more detailed level in the later sections).
- What terms need to be defined to convey the main points to the intended audience.

Intended audience For the project reports for this course, the intended audience is *not* the course instructor.

- Aim for a broader audience such as the professional programming community, or, more likely, limited to C++ programmers (I leave some discretion here, but you should make clear how much knowledge of programming, C++, STL or BGL you assume).

- Thus the document should *not* have a title like "Advanced Programming Project Report"; instead it should be descriptive of the library component(s) developed in the project.

- The documentation should *not* refer to earlier work on the project such as preliminary or progress reports; most readers won't care about the history of the project. You might include some information such as results of preliminary experiments with compiler portability or performance, but only if it helps justify design decisions of your final software (so it probably goes in the design document if it is included at all).

Level of discussion This is one of the most difficult problems of technical writing; it is tempting to dive into low level details in the overview long before the reader is ready for them. This can also be a problem in tutorial material (if any); you must introduce details a few at a time or else the reader will be confused.

One crucial issue is how much, if any, to say about design decisions in the overview. As a general rule, discuss design decisions in the overview only to the extent you believe that doing so will help convince your audience it is worth trying your software. Overviews of STL, for example, generally include discussion of many design decisions as part of the explanation of the advantages of generic programming.

Defining terms This will depend on your intended audience; the more background you assume, the more you restrict your audience, but spending a lot of space defining terms may turn off readers who are already knowledgeable.

Tutorial material After the overview, the User's Guide often contains tutorial material of various forms, particularly examples of use of the software and discussions of pitfalls to avoid. Writing good tutorials is important but is mostly beyond the scope of this project. Spend time on this part only if you have everything else completed and polished.

Reference manual In How to use the STL documentation[48] note the descriptions of the format of concept pages, of type pages, and of function pages. You should take these as guidelines for writing the reference manual part of the documentation for your project.

Design issues In this section describe the key design decisions. This includes interface and abstract semantics design decisions, and why particular algorithms and data structures were chosen, versus alternatives. Remember that this documentation is not for the end-user, and thus may be more technical than the User's Guide.

As much as possible, separate the design issues into an abstraction level and an implementation level:

[48] http://www.sgi.com/tech/stl/doc_introduction.html

- The *abstraction level design issues* concern the interface, semantics, and performance requirements.
- The *implementation level design issues* concern the choice of internal algorithms and data structures that meet the abstraction level requirements. Do not discuss details that would be obvious from reading the code, but if you have used nonobvious optimizations in your code then either here or the next section (source code) is the place to describe them.

Occasionally you may need to refer to some of the implementation issues in justifying your abstraction level design choices.

Some further points:

- It may be useful to make comparisons to other libraries or software packages. Though not essential, such comparisons can help readers who might have knowledge of other software put your library in perspective. Avoid comparisons to software that is not widely known, since only a few people would be able to appreciate the comparison. Also avoid denigrating competing software or design decisions with phrases like "an awful hack"; keep your comparisons on a technical basis.
- Give a detailed rationale for your design decisions if you think it will either (1) help convince readers to use your software, or (2) help future maintainers, re-implementors, or extenders of your software to avoid making other design choices that you know are inferior (because you tried them, or rejected them on the basis of careful analysis).
- By the same token, avoid rationalizing decisions for which you have little supporting evidence; it's better simply to say that other decisions were possible but were not explored.

Source code As mentioned above, you are required to present the complete source code using the literate programming style. In some cases you may find it easier to present design decisions and source code if you combine them in a single section. This is permitted.

Keep in mind the flexibility the literate programming style permits: you can present the most important parts of your code first, followed by the details, assembling the parts into an order acceptable to the compiler in an "overview" or "outline" part (which can come either before or after the other parts). You may even relegate nitty-gritty details to an appendix so they don't make the main discussion too long.

Test plan and results As mentioned in the initial project assignment, testing should be extensive, including checking correctness of small cases, including boundary cases such as empty sequences, and measuring performance of large randomly generated inputs of a variety of sizes (possibly up to 100,000 or a million elements, depending on the algorithms or data structures involved).

- Correctness for large inputs should also be checked, using an acceptance routine. For example, to check a sort routine, use an acceptance routine that checks to see if the result is in order, or better, checks to see if it is exactly the same output as that from another sort routine known to be correct. For this to work, the benchmark routine must have the same stability properties as the routine under test.
- In timing fast algorithms on moderate size inputs (a few thousand elements) you may need to repeat some number of trials of the algorithm, record the total time, and divide that by the number of trials, to get an accurate average time. Note that this is different from (and usually will be more accurate than) timing each trial, totaling the times, and dividing by the number of trials.
- Be sure to say which platforms you have run your tests on, giving the hardware platform (including processor speed) and compiler (including version and option selections, especially the optimization level). If you have run tests with more than one platform/compiler combination, make clear which of your test results correspond to which combination.

Other essential steps in preparing your report

1. Pay attention to grammar, spelling, and punctuation. Any problems in these areas can distract readers from your main message, or, worse, drive them away entirely.
   (a) Read, or reread, Strunk and White, *The Elements of Style*. It's short, pleasant to read, and always helpful.
   (b) *Always* run your documents through a spelling checker before submitting them to anyone else to read.
   (c) Remember also to check manually for problems that a spelling checker can't detect (see below).
   (d) Often it's difficult to see grammatical errors in your own writing, even if you know the rules, so have someone else proofread it too.
2. Document formatting errors can also be distracting. See the LaTeX section for help in avoiding common errors.
3. Write clearly and simply, and maintain a level of formality of style. In a team project, different parts of the report will typically be drafted by different members of the team, with inevitable differences in style. One team member should take the role of editor, resolving such differences to achieve a single writing style in the final document.

Ode To My Spell Chequer
Author Unknown

I have a spelling chequer,
It came with my PC.

It plainly marks for my revue,
Miss takes I can not sea.

I've run this poem threw it,
I'm sure your please too no.
Its letter perfect in it's weigh,
My chequer tolled me sew.