

Final Project Report
ACS Teaching with Technology Fellowship

*HTML Generator
For
Instructor-Annotated Computer Program Source Listings*

Richard H. James
Rollins College

Abstract:

Would providing computer science students with interactive, instructor-annotated example program listings enable students to deepen their knowledge outside of class? The Web provides the vehicle for providing annotated listings if an instructor will take the time to generate the HTML files.

A proven technique for learning and reasoning about software design is to examine computer program listings generated by experts. [OLA 00] The instructor's explanations and probing questions make this a useful classroom activity but how much information can the students capture for later study? If the example listings were annotated with the instructor's comments and presented interactively through a Web browser, then potentially students could study these programs at any time.

One of the learning objectives of the computer science curriculum is to develop computer programs that are structured for readability by others. This structure can be lost if a program's source code file is delivered as-is to a Web browser. Any Web page displaying example listings must convey the program's proper structure so that students can begin to assimilate this style.

This paper presents a software application that generates the HTML source files for an interactive Web page from an example computer program source file. This application allows instructors to produce Web pages for student use by simply adding their comments to the program's source file. The application generates HTML formatting tags that preserve the example listing's structure. The author developed this application after recognizing that manually producing a collection of Web pages with annotated programs for a typical computer science class required a considerable amount of time regardless of its pedagogical benefits.

Hypothesis:

Student outcomes in introductory computer science courses can be improved by providing Web pages of example program listings annotated with an instructor's comments. To enhance the learning experience, the example program must be displayed with its correct structure and the student must actively engage with the displayed material. Since several researchers have identified the downside of producing

"multimedia for educational purposes" as being the significant time required by the developer, a tool is needed to quickly generate the HTML files with minimal effort on the part of the instructor. [ADA 96]

Background:

In introductory as well as advanced computer science courses, a code "walk-through" assists students with learning specific programming design features, language syntax, and style. A code walk-through consists of displaying a computer program's source listing and discussing its important and topical features with the class. This is a standard industry technique used to both evaluate new programs and to help new software developers understand the style and designs used by an organization's senior developers. [BEI 96]

Discussing a program's source listing with the class creates a dynamic learning environment in the classroom. A code walk-through helps focus students' attention on specific computing topics and demonstrates how to use a specific program design technique as well as how to implement it using a software language's features. Using example program listings from the course's text allows the students to annotate their text during the discussion. Providing the students with a paper copy of the code using an enlarged font set gives them even more space to make notes. By projecting the source files from within a development system's project window (IDE), the instructor can both discuss the features of the program and execute the program to demonstrate its functionality. However, research into how the brain learns indicates that integration of new information into knowledge requires time.[ZUL 02] Time for reflection may not be adequate in the classroom. Students need an opportunity to review the example code *along with the instructor's insights and observations.*

If students do not fully comprehend the code walk-through's classroom discussion, then they must enhance their understanding by interpreting the code example, as well as the related programming topic, outside of class with only the text and their possibly poor or incomplete notes as guidance. Most authors of computer science texts do not provide exhaustive comments for their example code listings. Yet for computer science students, understanding the resultant action of each statement in an example program, or the benefits of a particular structure of a set of statements, is essential to deepening their learning of the design concept, the programming language and proper programming style.

It is known that students learn most of a course's concepts outside the classroom [DAV 94, WAL 98]. In addition, most students are more engaged in the learning process when they are actively doing something instead of just reading text - the concept of active learning [DAV 94, REI 97]. To assist computer science students to learn outside the classroom, a method was needed to allow them to repetitively review and study example source listings *with the instructor's comments included and with the proper structure preserved.* Allowing students to interact with the material through their Web browser provides them a "substantial degree of learner control - an important feature of multimedia applications" for learning enhancement. [ADA 96]

An obvious approach is to provide annotated program listings on the course's Web site. However, just copying a program's source listing into a simple HTML file does not completely aid students' learning – although it can be done quickly and easily. The



```
date.cpp
#include "date.h"
#include <time.h>
#include <stdio.h> // for sprintf

/*****
This code is freely distributable and modifiable providing you
leave this notice in it.
Copyright © Owen Astrachan
*****/

static int DaysInMonth(int, int); // # of days in month in year
static bool IsLeap(int year); // is year a leap year

static string dayNames [] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                             "Thursday", "Friday", "Saturday"};

static string monthNames [] = {"January", "February", "March", "April",
                               "May", "June", "July", "August",
                               "September", "October", "November", "December"};

Date::Date()
// postcondition: date initialized to default date (today)
{
    static struct tm timeHolder;
    static struct tm *date = &timeHolder;
    time_t tloc;

    time(&tloc);

    date = localtime(&tloc);
```

structure and format of a program's source listing assist humans in reading and understanding the code. Unfortunately direct copying of a source file's text into an HTML file loses the critical structure of the source code. Figure 1 presents a properly structured example as it would appear in a text.

Figure 1. Properly Formatted Source Listing from Text

Figure 2 is an image of the same program if presented as-is by a Web browser.

```
1 #include "date.h"
2 #include <time.h>
3 #include <stdio.h> // For sprintf
4
5 =====
6 This code is freely distributable and modifiable providing you
7 leave this notice in it.
8 Copyright © Owen Astrachan
9 =====
10
11 static int DaysInMonth(int, int), // # of days in month in year
12
13 static bool IsLeap(int year), // is year a leap year
14 static string dayNames [] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"},
15 static string monthNames [] = {"January", "February", "March", "April", "May", "June", "July", "August",
16 "September", "October", "November", "December"},
17 Date Date()
18 // postcondition: date initialized to default date (today)
19 {
20     static struct tm timeHolder;
21     static struct tm *date = &timeHolder;
22     time_t loc;
23
24     time(&loc);
25
26     date = localtime(&loc);
27 }
```

Figure 2. As-is Formatted Source Listing in Web Browser

Of course, “as-is” means just copying the text of the source listing into an HTML file and adding only the title and body tags as well as the individual end-of-line tags (to keep the text from appearing as one long continuous line). Even the as-is version requires some instructor time and effort.

Putting the example listings from the course’s text on a Web Site provides students nothing more than another block of text to read. Most publishers now provide a Web Site in support of a text, so students have the capability to download and review the code examples. The downloaded files can be opened within a development environment which preserves the listing’s structure. However, the instructor’s comments and observations that illuminated and highlighted specific features of the code are not available to the student while studying the program from either a simple Web page or their text editor outside of the classroom.

Research Analysis:

One objective of this research was to design and develop a software application to generate Web pages that incorporated instructor comments into example programs. Another objective was to preserve the structure of the example programs when displaying the generated Web page. A third objective was to provide interactive features to engage the student with the material. The final objective was to conduct pedagogical research to examine whether student outcomes were improved due to the use of annotated Web pages outside of the classroom. The initial hypothesis was that providing a means for computer science students to repetitively review example programs, with annotated comments explaining the details of why and how the code was designed, would deepen the students’ skills and knowledge. The generated Web pages would interactively provide students with their instructor’s insights to support their reflection on the topical concepts.

Web pages can be constructed to be responsive to user actions. Button images can be added to a page that, when clicked by the user, open a new browser window. The size of the new window can be specifically set so that it is reduced in size compared to the first browser window which remains open. Another responsive device on a web page can be the use of text boxes or selection buttons that present questions to the user. When answered, these are then submitted to the server. Dependent on the user's answer, a new window opens providing the user with feedback on their answer. These responsive techniques generate the common Web browser technique of a "pop-up" window – only under the control of the user.

A pop-up window, opened in response to the student's action, can display the instructor's detailed commentary or observations about either the adjacent program code statement(s), or about the answer submitted in response to a question. Keeping the pop-up window small allows the student to view both the example code on the original page as well as the instructor's comments in the pop-up window. The pop-up windows can be moved around the screen to allow the student to view all the lines of the example code referred to by the instructor's comments. If the pop-up information is in response to a student's answer to a question, the appropriate portion of the example listing can still be viewed to allow the student to better understand the response.

When the student is finished with the information in the pop-up window, they close that window and return their focus to the original Web page with the example listing.

An essential part of this work was to design a Web page format that would both preserve the structure of the example program's source listing and still allow for the insertion of the active devices that the student would use to access the instructor's comments. Experimental versions of the proposed Web-based technique were developed (see Figure 3) which can be viewed at (<http://web.rollins.edu/~rjames/research/cse/htmlmaker.html>). Creating these Web pages demonstrated that the structure of a program's listing could be preserved using standard HTML tags while instructor comments could be embedded using pop-up windows. These experiments also revealed that a considerable amount of time was required of the instructor to manually generate these Web pages. The author used the HotDog (Version 6) HTML editor to develop the experimental Web pages.

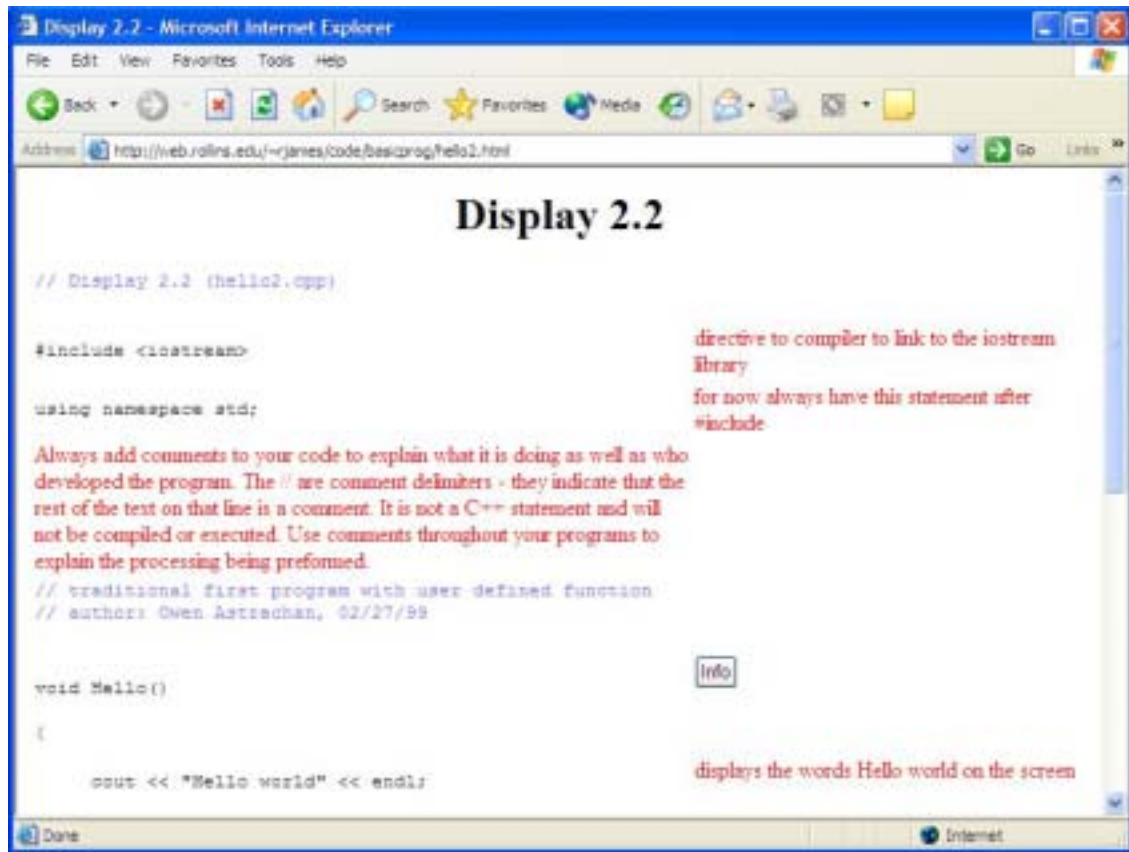


Figure 3. Proof-of-concept Web Page

Technical Approach:

The experimental web page development activity described above identified the set of HTML tag elements that allowed both the capture of the listing's structure as well as the inclusion of instructor annotations. Figure 4 shows the basic layout of the design of the HTML page's format.

HTML Page	
Title	
Left Column	Right Column
Source listing	Instructor's Annotations
Row x <code>#include <iostream.h></code>	<input type="button" value="Info"/>
Row x + 1 <code>int main ()</code> <code>{</code> <code>(indent) char aChar;</code>	Declaring a variable of type char.
.	
.	
.	
Row y <code>cout << "Done";</code> <code>} (return)</code>	

Figure 4. HTML Page Layout

The HTML page is divided into a two-column table. The left-column holds the original program listing including any comments included in the listing by the *developer of the program*. By examining the developer's comments as well as the associated code statements, students increase their knowledge of how to produce comments as a critical asset of a program's listing. The right column contains the button images that open (when clicked on) the pop-up windows displaying the instructor's observations about the adjacent program statement in the left column. The table's rows contain the example program's code statements in the left-hand cell (column) with the instructor's annotation in the right-hand cell (column). Use of individual table rows ensures alignment between the instructor's comment and the appropriate statement or block of statements from the example.

JavaScript code is embedded in the generated HTML page to create the pop-up windows that contain the instructor's information.

A design decision was made to provide the instructor with an additional method to include their annotations in the Web page. This second method allows the inclusion of brief instructor comments as text in the right-hand cell of a row. This feature minimizes use of the network's bandwidth for downloading the additional HTML page for a pop-up window which may provide only a small set of information. Instructors can use either annotation method or a combination of both throughout a Web page. To aid the student's understanding, the font of any developer comments, from the original source listing (displayed in the left-hand column), is colored blue while any instructor's annotations in the right-hand column are in red.

Correct indenting of the program's listing is required to preserve its structure. The indenting is implemented by using the HTML format element `` (unordered list) for each indent toward the right margin and the `` element (close unordered list) for each subsequent return toward the left margin. These elements can be nested to allow nested indents toward the right margin as well as repeated returns toward the left margin of the page.

The font of the program listing text (in the left-column) is set using the HTML text formatting element `<code>` to provide the reader with the typical non-proportional font of a program listing. The text of all instructor annotations (either in the right-column or in the pop-up windows) employs the browser's default proportional text font.

The Web pages containing the annotated example listings can then be linked to the course's web site to keep pace with the classroom code walk-through activities as well as the text reading assignments. As the course progresses, the students have the benefit of being able to go back and review or reflect on any previously posted annotated listings. The instructor can also go back and revise or add to the annotated comments to highlight features in any example program that students are still using incorrectly in their lab and programming assignments.

While studying, the students now have the capability to actively review any text example with the instructor's comments available. When the student shapes a mental question about an example listing, the Web page provides an opportunity for the student to obtain the instructor's comment and to their own answer. Deep learning occurs when students generate their own questions during reflection on the material and then search for the answer.

An added benefit of this Web presentation methodology is that the set of annotated code examples for introductory courses can be linked to an index page on the Web site. Students in future courses that need to review a particular language feature or style can use the index page to quickly locate an annotated example that answers their question. The annotated examples index page can be made accessible from the Department's or instructor's Web Site for use by all current students. Annotated example Web pages could also be shared between instructors to minimize the generation effort. Repositories could be set up to hold these pages for a broader range of use among CS instructors.

Assessment:

The Web presentation technique described in this paper has been minimally used during the past two terms (Fall '02 and Fall '03) in our Introduction to Computing course, CMS 167. The technique is being used this term (Spring '04) in the Operating Systems Design Principles course, CMS 370. A review and analysis of students' work from previous CMS 167 courses is being conducted to identify additional topic areas that introductory students find difficult to master. At the completion of the CMS 167 course in the Fall '04 term during which the Web presentation of the annotated programs will be used extensively in support of the course, a comparison of student outcomes will be made. The comparison will include both exam scores as well as programming and lab assignment evaluations. It is hypothesized that student performance will be enhanced due to the use of the Web presentation technique.

In addition to the student outcomes comparison described above, a survey instrument will be prepared and administered to the Fall '04 CMS 167 and Spring '04 CMS 370 students to elicit how they used and felt about the new learning technique. Based on the performance comparison and the survey, a determination will be made to continue or discontinue this Web based technique; and, if continuing, how improvements can be made.

Technology Transfer/Future Work:

Displaying an example program's listing in its original structure on any Web browser with any screen resolution necessitates the use of some complex Web page formatting techniques. In addition, the technique for including the "pop-up" windows could be considered difficult by instructors not familiar with certain HTML coding techniques. To ease the burden on the instructor, a software application was developed that takes a text source file as input and produces the annotated Web page. A set of delimiters were designed for the instructor to use to identify the parts of the source file that are the annotation comments that need to appear in either a "pop-up" window or in the right-hand column using red font. The application generates both the base HTML page with proper structuring of the program's listing as well as all the HTML pages that "pop-up" with the instructor's annotations. The new software application allows this learning technique to be easily used by my colleagues here at Rollins and at other ACS institutions.

Appendix A contains the details of how to use the HTML generator software application as well as links to the Web Site that contains the author's annotated code.

The current application has been tested with both C and C++ source listings produced by using the CodeWarrior Development Environment. The application was also tested against source code provided with Astrachan's text. [OLA 00] The author will be testing the application with Java source listings during the Spring '04 academic term. Revisions may be made due to suggestions from student surveys. The author is also preparing a port of the application for both Macintosh and UNIX platforms.

If this Web-based teaching technique proves successful, the HTML generation application could be modified for any course that uses instructor's annotated examples as a teaching method. Mathematical proofs, chemical analyses, foreign language translation (most students feel programming languages are foreign), and literature passages are just a few of the possible disciplines that could benefit from this application of Web technology. Critical to its acceptance by other disciplines is the software application that minimizes the Web page development burden on other instructors.

Availability:

The current executable file for the HTML Maker can be obtained from the author's site:

<http://web.rollins.edu/~rjames/research/cse/htmlmaker.html>.

References:

- ADA 96 Adams, Elizabeth S., *et.al.*, "Interactive Multimedia Pedagogies" in *Proceedings of the Integrating Technology into Computer Science Education Conference*, Association for Computing Machines, 1996.
- BEI 96 Beizer, Boris, *Software System Testing and Quality Assurance*, International Thomson Computer Press, Boston, MA., 1996
- DAV 96 Davidson, Cliff I. and Susan A. Ambrose, *The New Professor's Handbook*, Anker Publishing Company, Bolton, MA., 1994
- REI 97 Reis, Richard M., *Tomorrow's Professor: Preparing for Academic Careers in Science and Engineering*, IEEE Press, Piscataway, NJ., 1997
- WAL 98 Walvoord, Barbara E. and Virginia Johnson Anderson, *Effective Grading: A Tool for Learning and Assessment*, Jossey-Bass Publishers, San Francisco, CA., 1998
- OLA 00 Astrachan, Owen L., *A Computer Science Tapestry*, McGraw-Hill, Boston, MA, 2000
- ZUL02 Zull, James E., *The Art of Changing the Brain – Enriching the Practice of Teaching by Exploring the Biology of Learning*, Stylus Publishing, Sterling, VA, 2002

Appendix A - Using HTML Maker

The executable file for the current version (1.x) of the HTML Maker application can be downloaded from the author's site (<http://web.rollins.edu/~rjames/research/cse/htmlmaker.html>). The application is currently compiled for WinTEL machines. Ports for Macintosh and UNIX machines will be coming soon.

Currently, the application has a simple (minimal) text-based user input. The user is asked for the following information:

- the relative path to the directory of the annotated source file
- the name for the base HTML file
- if the source file was generated using a Macintosh development system

The generated HTML files (including the base and the pop-up files) will be placed in same directory as the annotated source file.

To prepare the example's source file for conversion to HTML, the instructor must use the following set of special annotation string delimiters. These delimiters were created within standard C/Java comment delimiters so that the source file will still compile after the annotations are added.

`/*^ ... ^*/` : used to identify the title of the HTML page. All pages must have a title. Use `/*^` just before the beginning of the text of the title; then, use `^*/` to end the string. You do not use the `...'`s .

`/*@*/` : used to identify the beginning of the statement in the listing that the annotation applies to. Note that this is a single delimiter that cannot contain any intervening text or spaces.

`/*# ... #*/` : used to indicate a brief annotation that will appear as text in the right-hand cell in red text. Include the information between the `/*#` and `#*/` (do not use the `...'`s).

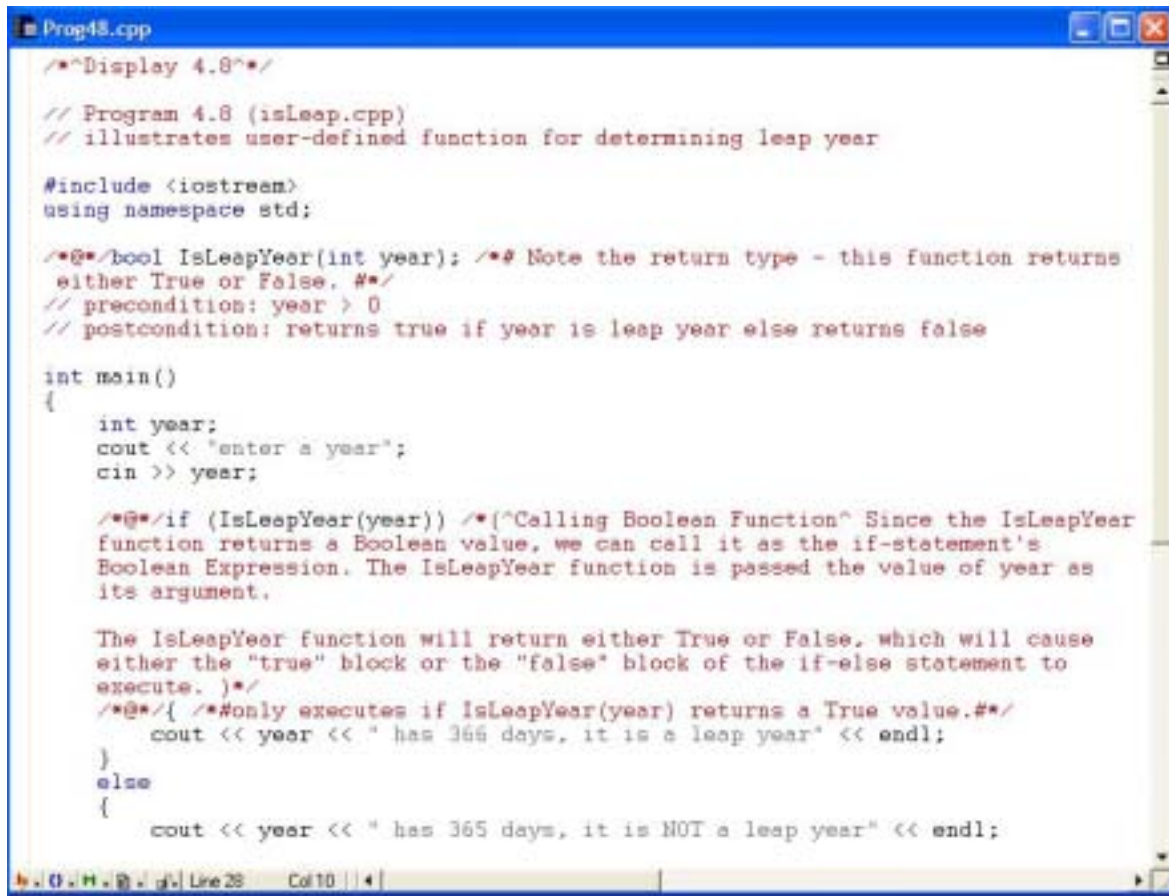
`/*(...)*/` : used to indicate an annotation that will appear in a pop-up window.

`^ ... ^` : Used inside of the above delimiter to indicate the title of the pop-up window.

Use of the standard C/C++/Java comment delimiters (`//` or `/*... */`) will cause the comment statement to appear in the left-hand cell with its text in blue font. These comments are from the source listing's designer.

Each opening curly brace (`{`) will cause an indentation of all following statements 4 spaces toward the right margin of the displayed page. Subsequent opening curly braces will cause an additional 4-space indentation to the right. Each closing curly brace (`}`) will cause a return of 4 spaces toward the left margin of all following statements. Subsequent closing curly braces will cause additional 4-space returns to the left.

Figure A.1 presents an example of an annotated source file using the annotation string delimiters discussed previously.



```
/*~Display 4.8~*/  
  
// Program 4.8 (isLeap.cpp)  
// illustrates user-defined function for determining leap year  
  
#include <iostream>  
using namespace std;  
  
/*~*/bool IsLeapYear(int year); /*~# Note the return type - this function returns  
either True or False. #~*/  
// precondition: year > 0  
// postcondition: returns true if year is leap year else returns false  
  
int main()  
{  
    int year;  
    cout << "enter a year";  
    cin >> year;  
  
    /*~*/if (IsLeapYear(year)) /*~(^Calling Boolean Function^ Since the IsLeapYear  
function returns a Boolean value, we can call it as the if-statement's  
Boolean Expression. The IsLeapYear function is passed the value of year as  
its argument.  
  
The IsLeapYear function will return either True or False, which will cause  
either the "true" block or the "false" block of the if-else statement to  
execute. )~*/  
    /*~*/{ /*~#only executes if IsLeapYear(year) returns a True value.#~*/  
        cout << year << " has 366 days, it is a leap year" << endl;  
    }  
    else  
    {  
        cout << year << " has 365 days, it is NOT a leap year" << endl;  
    }  
}
```

Figure A1. Example of Annotated Source File

Figure A.2 shows the resultant HTML file in a browser window for the annotated source file, shown in Figure A.1.



Figure A.2. Browser Display of Source Listing from Figure A.1

06/09/04