

Laboratory Manual With Lecture Notes

for use with

C++ Program Design An Introduction to Object-Oriented Design

Third Edition

James A. Cohoon
University of Virginia

Jack W. Davidson
University of Virginia



Boston Burr Ridge, IL Dubuque, IA Madison, WI New York
San Francisco St. Louis Bangkok Bogotá Caracas Kuala Lumpur
Lisbon London Madrid Mexico City Milan Montreal New Delhi
Santiago Seoul Singapore Sydney Taipei Toronto

McGraw-Hill Higher Education 
A Division of The McGraw-Hill Companies

Laboratory Manual with Lecture Notes for use with
C++ PROGRAM DESIGN: AN INTRODUCTION TO OBJECT-ORIENTED DESIGN, THIRD EDITION
JAMES A. COHOON AND JACK W. DAVIDSON

Published by McGraw-Hill Higher Education, an imprint of The McGraw-Hill Companies, Inc.,
1221 Avenue of the Americas, New York, NY 10020. Copyright © The McGraw-Hill Companies, Inc., 2002, 1998.
All rights reserved.

No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system,
without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, network or other electronic
storage or transmission, or broadcast for distance learning.

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 QPD/QPD 0 3 2 1

ISBN 0-07-251859-6

www.mhhe.com

General manager: Tom Casson
Publisher: Elizabeth A. Jones
Senior developmental editor: Kelley Butcher
Marketing manager: John Wannemacher
Senior project manager: Kay Brimeyer
Production supervisor: Enboqe Chong



Contents

Preface, v
Slide set 1
<i>Basics of machine, software, and program design, S1</i>
Slide set 2
<i>Fundamentals of C++, S16</i>
Slide set 3
<i>Modifying objects, S32</i>
Slide set 4
<i>If constructs, S55</i>
Slide set 5
<i>Iterative constructs, S69</i>
Slide set 6
<i>Libraries, S103</i>
Slide set 7
<i>Programmer-defined functions, S114</i>
Slide set 8
<i>Class construct, S139</i>
Slide set 9
<i>Abstract data types, S148</i>
Slide set 10
<i>Arrays, S171</i>
Slide set 11
<i>Vectors, S189</i>
Slide set 12
<i>EzWindows API, S213</i>

Slide set 13	
	<i>Pointers and dynamic objects, S226</i>
Slide set 14	
	<i>Inheritance, S248</i>
Slide set 15	
	<i>Templates and polymorphism, S265</i>
Laboratory 1	
	<i>Riding the wave of the future, 1</i>
Laboratory 2	
	<i>Attacking your first problem, 9</i>
Laboratory 3	
	<i>Inquiring minds want to know about the if statement, 17</i>
Laboratory 4	
	<i>Let's go looping now, everybody is learning how, 25</i>
Laboratory 5	
	<i>Taking a trip to the library, 37</i>
Laboratory 6	
	<i>Pass it on, 43</i>
Laboratory 7	
	<i>Functional living, 57</i>
Laboratory 8	
	<i>Getting classy, 69</i>
Laboratory 9	
	<i>Now that's classy, 77</i>
Laboratory 10	
	<i>EzWindows and event-based programming, 83</i>
Laboratory 11	
	<i>Hurray for arrays, 93</i>
Laboratory 12	
	<i>Vectoring in on vectors, 103</i>
Laboratory 13	
	<i>Inheritance, 113</i>
Laboratory 14	
	<i>So far so good, 121</i>
Appendix A	
	<i>EzWindows API reference manual, 127</i>



Preface

INTRODUCTION

This revised laboratory manual and accompanying lecture slides are designed to be used in conjunction with a C++ introductory text providing broad integrated coverage of the ANSI C++ standard and its Standard Template Library (STL). In particular, the manual is a companion to the text *C++ Program Design: An Introduction to Programming and Object-Oriented Design 3rd Edition*. We have also revised our EzWindows Graphical API in response to user feedback.

Unlike most laboratory manuals that are designed to be self-study aids for mastering syntax or to have students perform straightforward, self-guided activities in “open” laboratories at a time and place of the students’ choosing, this laboratory manual is designed for use in a “closed” laboratory. A closed laboratory meets at an assigned time and place with a laboratory instructor, and, depending upon class size, laboratory assistants. Each closed laboratory activity typically illustrates concepts from lecture by using examples, implementations, and problems that are designed to challenge the student. A closed laboratory environment provides a student with the opportunity to try various options or approaches and to receive immediate feedback. Similarly, when mistakes are made or clarification is needed, help is immediately available.

We have used closed laboratories in our introductory computer science course for the past nine years, and our department has been extremely pleased with the results. Standardized tests show that on average our students leave the course with a much better mastery of the material than they did prior to our switch to closed laboratories. Additionally, student evaluations show that they believe the laboratory activities are a major contributor to their understanding of course material. Students cite the laboratories as the most interesting, useful, and fun part of the course. Another indicator of the positive effect of the closed laboratory is the large number of students who volunteer as laboratory assistants in subsequent offerings of the class.

USING THIS MANUAL

In each week of the course we teach there are two lectures followed by a two-hour laboratory. Each laboratory activity in this manual is designed to illustrate and explore concepts from that week's lectures. Thus, the pace of the course is very much driven by the laboratory activities.

Our typical laboratory has close to forty-five students, with two supervising laboratory instructors and two undergraduate teaching assistants. Every student has a computer, but we have found that having students work together in small groups of two or three is often quite useful—students help each other and share the responsibility of teaching and learning. We arrange students to have different partners each week, and encourage them to seek assistance from a laboratory instructor at any point during the laboratory if they need help.

Individual activities are not graded but attendance is recorded and is a factor in the final grade. We purposely do not grade individual activities because we view the laboratories as a chance to explore and learn without time or grade pressures. However, some of the laboratories are designed so that an average student is hard-pressed to finish in the allotted time. Students who do not complete laboratories are encouraged to finish them independently.

Each closed laboratory consists of a set of experiments to be performed by the student. This model is very much like the familiar closed laboratories of chemistry or physics courses. For each laboratory there is a write up, a check-off sheet, and an experimental “apparatus.” A tear-out check-off sheet for each laboratory can be found at the back of this manual. The experimental apparatus is a set of programs and data files that the student sets up at the beginning of each laboratory by obtaining the appropriate files and loading them onto the computer. These files are contained in a self-extracting archive whose location is specified by the laboratory instructor.

Versions of the self-extracting archives for several popular compilers can be found by visiting our home page

<http://www.mhhe.com/c++programdesign/>

At various points throughout the laboratory, students are required to demonstrate some code, answer a question, or explain some behavior that they have observed. These points are indicated by the



symbol in the margin and a ✓ in the text. Depending on the circumstances, the student is asked to write an answer in a boxed area or to simply give the answer to a laboratory instructor. When a student reaches one of these “check-off” points and believes he or she is ready, the student signals a laboratory instructor and gives her or his answer. Depending on the type of question, the response, and the student involved, the laboratory instructor may simply initial the corresponding entry on the laboratory check-off sheet and proceed to the next student who requests assistance. If the response is incorrect or incomplete, the laboratory instructor may help until the student is comfortable with the concept being explored. For some motivated students, the instructor may suggest an

additional experiment. For those questions for which there is no right or wrong solution, the laboratory instructor may explore other solutions with the student to further reinforce comprehension. When students complete the laboratory, they turn in their check-off sheets to the laboratory instructor to serve as a record of laboratory progress.

Many of the laboratories make use of a graphical Application Programmer Interface (API) designed specifically for beginning programmers to develop interesting programs. We provide a portable, object-oriented graphical library, named EzWindows, for the easy display of simple geometric, bitmap, and text objects. Using the API provides several important experiences for the student. First, students are client users of a software library. Using well-designed objects helps students to appreciate good object-oriented design. Their experience as users forms the basis for becoming designers. Second, the API introduces students to the real-world practice of developing programs using an application-specific library. Third, using EzWindows to perform graphical input and output exposes the student to event-based programming and the dominant mode of input and output used in real applications, and it permits development of exciting and visually interesting programs. This experience motivates the students, and it provides a visually concrete set of objects that help students understand the object-oriented paradigm. EzWindows is simple enough that it allows even the first programming assignments to be graphical. A complete description of the EzWindows library is provided in the appendix.

LABORATORY SUMMARIES

- *Laboratory 1: Riding the wave of the future.* This introductory laboratory teaches students the basic skills that they will need to complete future laboratories. These skills include copying files, deleting files, backing up files to a floppy disk, creating directories, compiling C++ programs, executing C++ programs, and accessing the C++ compiler's on-line help facility.
- *Laboratory 2: Attacking your first problem.* In this laboratory students are guided through the process of decomposing a problem into steps and then translating those steps into working C++ code. The laboratory also exercises basic C++ programming skills that have been introduced in the lectures. Students practice input and output operations using the `iostream` objects `cin` and `cout` and also translating mathematical formulas into C++ assignment statements. In addition, the laboratory guides the students through the creation of a project file that uses the EzWindows API and some of its graphical objects.
- *Laboratory 3: Inquiring minds want to know about the if statement.* The objective of this laboratory is to ensure that students have a good understanding of the operation and use of the `if` statement. This laboratory also introduces the concepts of syntax and logic errors and differentiates between them. Students learn the practical skill of how to use a debugger to find and fix logic errors. The laboratory also continues to develop the

student's familiarity with object-oriented programming by using some of graphical objects found in the EzWindows library.

- *Laboratory 4: Let's go looping now, everybody is learning how.* In this laboratory the students explore two C++ looping constructs—the `while` and `for` statements. In addition to teaching students how to use and write looping constructs, the laboratory teaches students about common looping problems, such as infinite loops, off-by-one loops, improper initialization of a loop counter, and incorrect termination conditions. This laboratory also reinforces the very important skill of reading a stream of data from a file. The laboratory concludes by having the student finish a program that uses nested loops to construct a complex geometric pattern using the EzWindows library.
- *Laboratory 5: Taking a trip to the library.* An important component of becoming a productive, proficient programmer in a programming language is to learn the facilities and capabilities that are offered by the libraries of that language. This laboratory introduces students to some of the facilities and capabilities provided by the standard C++ class `string`. The laboratory also strengthens student understanding of how to use and manipulate objects with nontrivial attributes and behaviors.
- *Laboratory 6: Pass it on.* This laboratory begins an in-depth exploration of function invocation. The focus of this laboratory is C++'s parameter passing mechanisms. Through numerous examples the laboratory has the student explore value and reference parameter passing mechanisms. After completing the laboratory, the student will have a strong understanding of C++'s parameter passing mechanisms.
- *Laboratory 7: Functional living.* The exploration of functions continues with this laboratory. Through numerous examples the laboratory reinforces and refines the student's knowledge of scope and name reuse. This laboratory also explores recursion by carefully examining the execution of a factorial program. The laboratory concludes by guiding the student through the development of a text-processing program that involves implementing various utility functions.
- *Laboratory 8: Getting classy.* In the preceding laboratories students have used objects from both the standard C++ libraries (e.g., `cin`, `cout`, `string`) and the EzWindows API (`RectangleShape` and `SimpleWindow`), but they have not defined their own class types. This laboratory begins the students' exploration of the class construct by examining a class that they have used in many of the previous laboratories—`RectangleShape`. The laboratory explores the fundamental concepts of a class such as public, protected, and private members, inspectors, mutators, and facilitators. The laboratory concludes by having the students develop a class to represent a line segment in three-dimensional space.
- *Laboratory 9: Now that's classy.* This laboratory continues the exploration of C++'s class construct. In this laboratory, an abstract data type `Rational` is extended in several ways. The student modifies class

Rational so that the rational number is maintained in a reduced form, and the student adds comparison operators to the class by overloading the operators ==, <, and >.

- *Laboratory 10: EzWindows and event-based programming.* In this laboratory, the student explores more of the capabilities of the EzWindows API. The student examines the concepts and mechanics of event-based programming by developing programs that use both the mouse for input and timers to control when actions take place. This laboratory also introduces how to load and display graphical images called bitmaps.
- *Laboratory 11: Hurray for arrays.* This laboratory develops the ability to use and manipulate arrays. Programs that contain common array manipulation errors are examined. This laboratory also introduces the activity of searching a list for a key value by examining, modifying, and running programs that use several different search techniques. The student performs an experiment that measures the efficiency of these search techniques.
- *Laboratory 12: Vectoring in on vectors.* This laboratory develops the ability to use and manipulate lists using the container vector from the Standard Template Library (STL). Several programs using the principal member functions of the STL are examined and developed. Lab coverage includes the use of both subscripts and iterators. The student also develops an implementation of the MergeSort sorting algorithm.
- *Laboratory 13: Inheritance.* Laboratory 13 examines C++'s inheritance mechanism. To explore the concepts and mechanics of inheritance, the student creates a new graphical shape called BoxShape. To illustrate how inheritance supports reuse, the class BoxShape is derived from the familiar RectangleShape class. At the conclusion of the laboratory, the student has the ability to extend existing C++ classes via single inheritance.
- *Laboratory Review: So far so good.* This laboratory reviews the skills developed in the first several laboratories by requiring the student to develop several small programs. We use this lab for the students to prepare for the mid term. Each program focuses on a programming skill that students should now be able to perform on their own. The featured skills are prompting for and extracting input, translating mathematical formulas to C++ code, checking the validity of input according to some stated criteria, writing a function that accepts optional parameters, opening a data file and processing the data, and using the EzWindows API to create a simple display according to a given specification.

The instructor has some flexibility in deciding how to use the laboratories. Much of Laboratory 1 should be review material for many students. If desired, Laboratories 1 and 2 can be combined into a single laboratory. For curricula with a required course that covers the skills developed in Laboratory 1, this laboratory can also be deleted. A review laboratory for the first half of the course is also included. Laboratory 10 covers the EzWindows API and it can be moved to later in the course. We often cover the EzWindows API before arrays

and vector and inheritance because students like to use it in their final programming project.

THE AUTHORS

Jim Cohoon is a professor in the computer science department at the University of Virginia and is a former member of the technical staff at AT&T Bell Laboratories. He joined the faculty after receiving his Ph.D. from the University of Minnesota. He has been nominated twice by the department for the university's best-teaching award. In 1994, Professor Cohoon was awarded a Fulbright Fellowship to Germany, where he lectured on C++ and software engineering. Professor Cohoon's research interests include algorithms, computer-aided design of electronic systems, optimization strategies, and computer science education. He is the author of more than 60 papers in these fields. He is a member of the Association of Computing Machinery (ACM), the ACM Special Interest Group on Design Automation (SIGDA), the ACM Special Interest Group on Computer Science Education (SIGCSE), the Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Circuits and Systems Society. He is a member of the ACM Publications and SIG Boards and is past chair of SIGDA. He can be reached at cohoon@virginia.edu. His Web homepage is <http://www.cs.virginia.edu/cohoon>.

Jack Davidson is also a professor in the computer science department at the University of Virginia. He joined the faculty after receiving his Ph.D. from the University of Arizona. Professor Davidson has received NCR's Faculty Innovation Award for innovation in teaching. Professor Davidson's research interests include compilers, computer architecture, systems software, and computer science education. He is the author of more than 60 papers in these fields. He is a member of the ACM, the ACM Special Interest Group on Programming Languages (SIGPLAN), the ACM Special Interest Group on Computer Architecture (SIGARCH), SIGCSE, the IEEE, and the IEEE Computer Society. He serves as an associate editor of *Transactions on Programming Languages and Systems*, ACM's flagship journal on programming languages and systems. He was chair of the 1998 Programming Language Design and Implementation Conference (PLDI '98). He can be reached at jwd@virginia.edu. His Web homepage is <http://www.cs.virginia.edu/~jwd>.

ACKNOWLEDGMENTS

We thank the University of Virginia and its department of Computer Science and the National Science Foundation for providing an environment that made this laboratory manual possible. Numerous colleagues, teaching assistants, and CS101 students have contributed to this laboratory manual. In particular, we would like to thank Alan Batson, Scott Briercheck, Tom Horton, John Karro, John Knight, Sean McCulloch, Jane Prey, Paul Reynolds, Stephen Wassell, Alfred Weaver, and Bill Wulf for their advice, comments, and suggestions.

We thank all of the people at McGraw-Hill for their efforts in making this project a reality. In particular, we thank: Tom Casson, for his support and encouragement; Sandra Hahn for her product management skills; Sandra Schnee for her project management skills; and John Wannemacher for his creative marketing ideas. Special thanks go to executive editor and publisher Betsy Jones and senior developmental editor Kelley Butcher for their unflagging support and efforts.

We thank our spouses Audrey and Joanne and our children for their efforts, cooperation, and sacrifices in making this book happen.

Finally, we thank the users of this book. We welcome your comments, suggestions, and ideas for improving this material. Please write in care of the publisher, McGraw-Hill, Inc., or send E-mail to cohoon@virginia.edu or jwd@virginia.edu.

J. P. C
J. W. D

LABORATORY 1

Riding the wave of the future

Objective

As we all know, computers are vital tools for solving problems in business, industry, or research. Understanding a computer's full capabilities better enables us to pursue whatever endeavor we choose. This laboratory manual is a tool that will allow you to experiment with computer science. As you progress through each laboratory, you may wonder how or why something works. The best way to discover the answer is to *try things out*.

Some of instructions in this lab might already be familiar to you. However, they are not familiar to everyone and our goal is to get everyone up to full speed.

Key Concepts

- Copying files
- Backing up files
- Creating directories
- Editing a C++ program
- Compiling a C++ program
- Executing a C++ program
- Accessing on-line help


1.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpp1ab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab01.exe`. The copy should be placed in the `cpp1ab` directory. Execute the copy to extract the files necessary for this laboratory.

1.2 FILE MANIPULATION

Let's learn how to make backups of files by copying `hello.cpp` from the `cpp1ab` directory to a directory on the floppy drive. Backing up files is a good idea in case something goes wrong and you want to go back to an older version of a file and start again.

- Put your floppy disk in the appropriate drive. On most machines, the floppy disk drive is called the `a:` drive. Throughout this laboratory, we will assume that `a:` is the name of the floppy disk drive. If your computer uses a different drive letter for the floppy disk drive, please use that letter instead of `a:`.
- You are going to create a directory on your disk. Directories are useful for organizing files. You might have a directory for programs, one for homework, and another for personal correspondence. This manual requires you create one or more directories to store the materials for each lab. For this lab you will create two directories in the root directory of your floppy. We will do this using the Windows Explorer. It can be found on most systems by clicking the Start button on the bottom tool bar. On some systems it might be necessary to then select the Programs submenu. Once you have found the Windows Explorer you can start it with a single click of the mouse.
- Scroll the left subwindow of Windows Explorer until you find the floppy disk icon and then click on the floppy drive icon.
- You want to create a new directory `lab01` on your floppy disk. Click once on the word File on the Windows Explorer's menu bar. Then select New by clicking on it. A submenu should appear. Next select the New Folders option. A directory initially named New Folder is created. Then supply the desired name `lab01`.
- Using the Windows Explorer, open the directory that contains the extracted laboratory files (the location is usually `c:\cpp1ab`). If you are unsure where you placed the archive `lab01.exe`, consult your laboratory instructor.
- Click once on the `hello.cpp` file in the Windows Explorer's listing of the desired directory.
- Click on the Edit menu and then select the Copy option.
- Return to directory `a:\lab01`.
- Click on the Edit and select the Paste option.
- Click on the copy of `hello.cpp` and then repeat the Edit Copy and Edit Paste actions to make a second copy of `hello.cpp`.
- Rename the second copy of `hello.cpp` to `greet.cpp`. You can do so by clicking on the file and then clicking on the File menu of Windows Explorer and then selecting Rename.

- Delete `hello.cpp` by clicking on the file and then selecting the Delete option from the File menu.
- When the computer asks for confirmation of the deletion, click on the Yes button.
- File `hello.cpp` no longer exists, but a copy `greet.cpp` does exist in the `lab01` directory of the floppy drive.
- Return to the root directory of your floppy disk by double-clicking on its icon in the left subwindow.
- To delete the `lab01` directory on the floppy disk, click on the icon for the `lab01` directory. Delete this directory the same way you deleted `hello.cpp`. Observe that the file `greet.cpp` is also deleted.
-  ▪ Show the laboratory instructor that you can make a new directory called `check` on the floppy drive and then delete it. ✓

1.3 THE INTEGRATED DEVELOPMENT ENVIRONMENT MANIPULATION

The compiler's Integrated Development Environment or IDE allows you to create and edit program and other text files, compile and run the programs, and perform debugging activities.

- If the compiler with its IDE isn't already running, start it up as described in your laboratory instructor's handout.
- Select the Open option from the File menu.
- In the Open a File window, update its Look In component to the `cpplab` directory. To change to this directory, select the down arrow with the mouse. A pull-down menu displays the available file systems for this machine. Choose the appropriate file system. For example, if you extracted the files to your hard disk drive, you would most likely choose `c:` from the pull-down menu. If you are unsure which drive to choose, ask your laboratory instructor.
- Check the Look In component again. Are you working in the `cpplab` directory? If so, proceed to the next step. If not, navigate through the file system by traversing (selecting) the directory path that leads down to `cpplab`.
- In the Open a File window, you should see a list of `cpp` files that were created when you executed `lab01.exe`. The file you want is `hello.cpp`. Click on this file name and then click on the Open button on the right side of the dialog box.
- The file will appear in its own window. You want this window to be the *active* window, so click on its title bar.
- Header information for the program should appear at the top of the file. Modify the heading to show your name, the lab section, today's date, etc.

If you click somewhere in this window, you should see a blinking horizontal line called the insertion point. When you start typing, whatever you type will appear here. You can move the insertion point with the mouse or the arrow keys.

- Enter the rest of the `hello.cpp` program exactly as it appears below along with the requested biographical information. You may need to click on this window to make it the active window that receives your keystrokes.

```
// Name:
// Student Id:
// Electronic Id:
// Lab section:
// Course section:
// Laboratory instructor:
// Date:
// Honor pledge:
// hello.cpp: displays a greeting multiple times
#include <iostream>
using namespace std;
int main (){
    for (int i = 0; i < 5; ++i) {
        cout << "Hello, World" << endl;
    }
    cout << endl << "Hit enter to finish" << endl;
    char reply;
    cin.get(reply);
    return 0;
}
```

- When you are done typing in the program, save your program by selecting the Save option from the File menu. If you do not save the program you can lose everything that you typed.
- Make sure that the `hello.cpp` window is still the active window. If so, you can compile and link your program by selecting the Build option from the Build menu. When the IDE asks whether a default workspace should be created, indicate yes. The Build operation will then produce a new window named Message. This window contains warning messages and errors.
- If you have problems compiling, compare the above program with the source line at which the compiler is signaling the error. Be sure that the lines are identical, including the punctuation! Sometimes it is necessary to check the preceding line for errant typing. After correcting the errors, if any, rebuild the program. If you cannot determine the error, consult the laboratory instructor.
- Make sure your program is the active window. Once you have a successful compilation, choose the Run option from the Debug menu. (The lightning icon also works). ✓



- To close the window that contains your program's output, click on the icon in the upper-left corner of the output window. The output window must be closed before the program can run again.

1.4 GETTING HELP

If you don't know or have trouble remembering the syntax of a certain C++ statement or function (which happens more often than some of us like to admit), the IDE provides a handy utility called *context-sensitive help*.

- Position your cursor over the keyword **for** in `hello.cpp` and click. Then press the function key F1.
- When you are done with that help file, make the `hello.cpp` window the active window and remove the semicolon from the end of the `cout` line. Build the program. The compilation will end with errors, and you should see various complaints about the syntax of your program in the Message window. Triple click on an error message and then press F1. You should see a help window that explains the error message.
- Finally, click on the word **namespace** in `hello.cpp` and perform the same action as you did before for getting context-sensitive help with the keyword **for**. You are now looking at the help topics for the keyword **namespace**. Select one of the context topics. In some of the topic discussions many of the words are color-coded to indicate that they are links to further information. Click on one such link.
- Close the `hello.cpp` workspace by selecting the Close Workspace option from the File menu.

1.5 MAKING PROGRAM MODIFICATIONS

- Select the File option on the menu bar and then select the Open option. Open the file `test1.cpp` from the `cpp1ab` directory.
- Build and run the program `test1.cpp`. (You will need to create a default workspace.)
- Follow the instructions on the output window and have fun.
- After the program runs, read the code and try to understand how it works.
- Personalize the program by adding your own `cout` insertion statements at the beginning and end of the program. Save your program and then rebuild it.
- Run the program to test it. Remember you need to close the output window before you rebuild the program. ✓
- Close the current workspace. However, do not close the IDE itself.



1.6 NOTHING IS PERFECT

When you run the next program, pay particular attention to the correctness of the results.

- Open the file `test2.cpp`.
- Examine the code. It translates an age in years to an age in hours, minutes, and seconds. Does anything look incorrect?
- Build the program and run the program. (You will need to create a default workspace).
- Unless you can read very quickly or have a very slow machine, the program executes too fast to view the output. We need an alternative way of running the program.
- Click on the Windows Start button. Then, on either the menu that pops up or the Programs submenu, click on the entry Command Prompt. (Sometimes the entry is named MSDOS Prompt).
- Change to the directory where IDE has placed your translated program and make that direct directory your current directory. On most systems it will be `C:\cpp1ab`. For other systems, it is often `C:\localdata`. The following commands make `C:\cpp1ab` the active directory.

```
C:
cd \cpp1ab
```

- Test the program by running it with the following ages: one year old, two years old, six years old, your age, and one hundred years old. Record your results in the following table. Depending upon the integer representation used by the IDE there might be a miscalculation. What would be this representation error? Show your laboratory instructor your results. ✓



Age (Years)	Age (Hours)	Age (Minutes)	Age (Seconds)
1			
2			
6			
Your age			
100			

1.7 FINISHING UP

- Select the Exit option from the IDE File menu.

- Copy any files you wish to keep to your own drive using the Windows Explorer.
- If you are instructed, use the Windows Explorer to delete the `cpp1ab` directory.
- If you are instructed, shutdown or reboot the machine.
- Hand in your check-off sheet.