Lab Manual

to accompany

C++ PROGRAM DESIGN

An Introduction to Programming and Object-Oriented Design

James P. Cohoon ■ Jack W. Davidson

with Lecture Notes

Lab Manual to accompany

C++ PROGRAM DESIGN

An Introduction to Programming and Object-Oriented Design

with Lecture Notes

James P. Cohoon

Jack W. Davidson

both from

University of Virginia

WCB McGraw-Hill

Boston, Massachusetts Burr Ridge, Illinois Dubuque, Iowa
Madison, Wisconsin New York, New York San Francisco, California St. Louis, Missouri

To Kathleen, Nina, and Terry

WCB/McGraw-Hill

A Division of ***McGraw-Hill*** Companies

LAB MANUAL TO ACCOMPANY C++ PROGRAM DESIGN: AN INTRODUCTION TO PROGRAMMING AND OBJECT-ORIENTED DESIGN

This book is printed on acid-free paper.

3 4 5 6 7 8 9 0 DOC/DOC 9 0 9 8

ISBN 0-256-24092-2

Editorial director: *Tom Casson*
Senior sponsoring editor: *Elizabeth A. Jones*
Senior developmental editor: *Kelley Butcher*
Marketing manager: *John Wannemacher*
Senior project manager: *Beth Cigler*
Production supervisor: *Charlene R. Perez*
Designer: *Michael Warrell*
Compositor: *Interactive Composition Corporation*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley & Sons Company*

http://www.mhcollege.com

Preface

INTRODUCTION

This laboratory manual and accompanying lecture slides are designed to be used in conjunction with a C++ introductory text. In particular, it is a companion to the text *C++ Program Design: An Introduction to Programming and Object-Oriented Design.*

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 five 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 one-hour 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 fifty 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 such 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/`

A floppy disk version can be obtained by contacting the publisher.

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: So far so good.* This laboratory reviews the skills developed in the previous laboratories by requiring the student to develop several small programs. 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.
- *Laboratory 9: 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 `Sim-`

pleWindow), but they have not defined their own class types. This laboratory begins the student's 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 10: Now that's classy.* Laboratory 10 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 11: The EzWindows API.* 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 12: 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 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.

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. Laboratory 8 is a review laboratory that can be omitted or replaced. Laboratory 11 covers the EzWindows API and it can be moved to later in the course. We cover the EzWindows API before arrays and inheritance because students like to use it in their final programming project.

THE AUTHORS

James P. 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, 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 over fifty papers in these fields. He is a member of the ACM, SIGCSE, SIGDA, IEEE, and the IEEE Circuits and Systems Society. He is currently chairperson of SIGDA and a member of ACM SIG Board and ACM Publications board. He can be reached at cohoon@virginia.edu. His WWW home page is http://www.cs.virginia.edu/~cohoon

Jack W. 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. In 1990, Davidson received an NCR 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 over fifty papers in these fields. He is a member of the ACM, SIGARCH, SIGCSE, SIGPLAN, 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 can be reached at jwd@virginia.edu. His WWW home page is http://www.cs.virginia.edu/~jwd.

ACKNOWLEDGMENTS

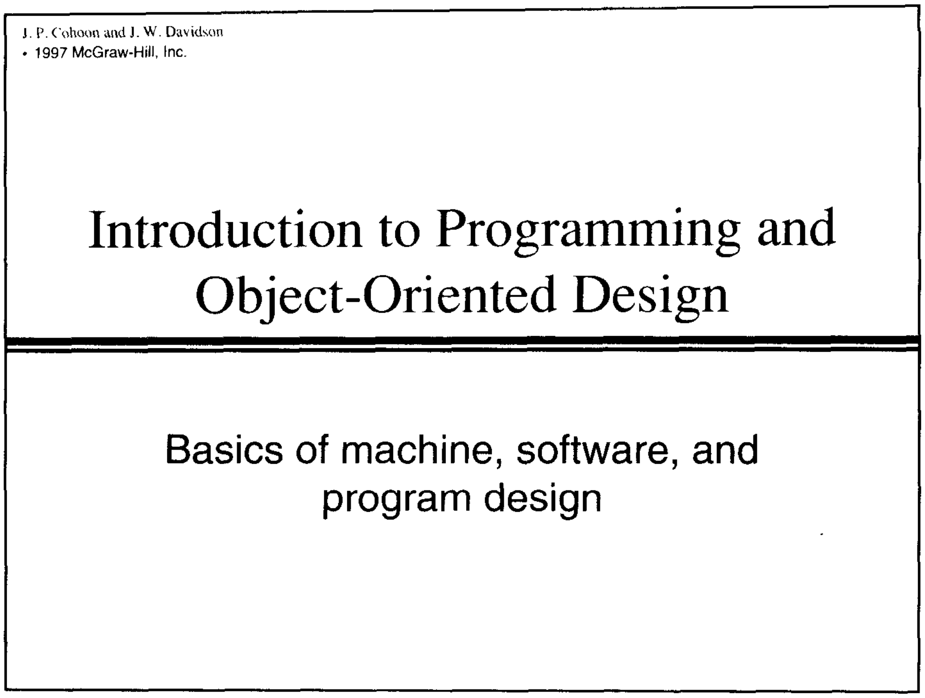
We thank the University of Virginia and its department of Computer Science and the National Science Foundation for providing an environment that made this laboratory manual possible. Numerous colleagues, teaching assistants, and CS101 students have contributed to this laboratory manual. In particular, we would like to thank Alan Batson, Scott Briercheck, Jane Prey, Paul Reynolds, 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; Kelley Butcher for her insightful analyses; Charlene Perez for her product management skills; John Wannemacher for his creative marketing ideas; Brad Kosirog, for his editorial coordination; and June Waldman, for her copy-editing ability. Special thanks go to executive editor Betsy Jones and senior project manager Beth Cigler 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.

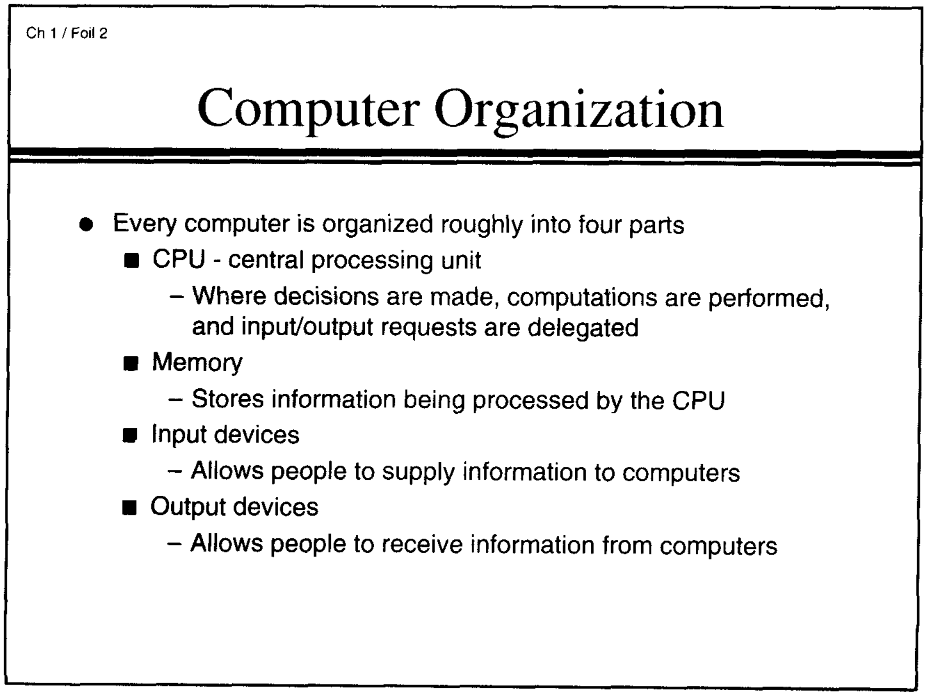
Finally, we thank the users of this book. We welcome your comments, suggestions, and ideas for improving this material. Please write in care of the publisher, McGraw-Hill, Inc., or send E-mail to cohoon@virginia.edu or jwd@virginia.edu.

J. P. C
J. W. D



Introduction to Programming and Object-Oriented Design

Basics of machine, software, and program design

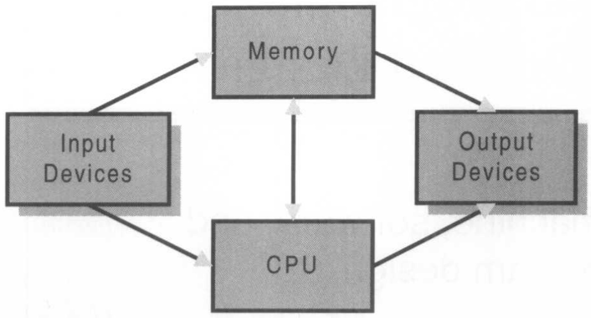


Computer Organization

- Every computer is organized roughly into four parts
  - CPU - central processing unit
    - Where decisions are made, computations are performed, and input/output requests are delegated
  - Memory
    - Stores information being processed by the CPU
  - Input devices
    - Allows people to supply information to computers
  - Output devices
    - Allows people to receive information from computers

Ch 1 / Foil 3

Computer Organization



Ch 1 / Foil 4

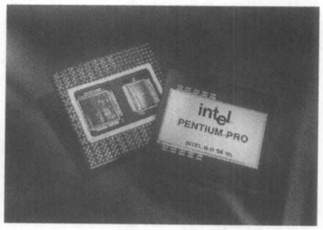
CPU

- "Brains" of the computer
  - Arithmetic calculations are performed using the Arithmetic/Logical Unit or ALU
  - Control unit decodes and executes instructions
- Arithmetic operations are performed using binary number system

Ch 1 / Foil 5

CPU

- Fundamental building block is a switch
  - Switches are made from ultrasmall transistors
- Examples
  - The Pentium ® processor contains about three million transistors
  - The Pentium Pro ® has about 5.5 million transistors



Ch 1 / Foil 6

Binary Arithmetic

- The individual digits of a binary number are referred to as bits
  - Each bit represents a power of two
- Examples

$$01011 = 0 \bullet 2^4 + 1 \bullet 2^3 + 0 \bullet 2^2 + 1 \bullet 2^1 + 1 \bullet 2^0 = 11$$

$$00010 = 0 \bullet 2^4 + 0 \bullet 2^3 + 0 \bullet 2^2 + 1 \bullet 2^1 + 0 \bullet 2^0 = 2$$

Binary addition $\longrightarrow$ $\begin{array}{r} 00010 \\ \underline{+\,01011} \\ 01101 \end{array}$ $\qquad$ $\begin{array}{r} 2 \\ \underline{+\,11} \\ 13 \end{array}$ $\longleftarrow$ Equivalent decimal addition

Binary Arithmetic

Binary multiplication

Equivalent decimal multiplication

```
     0101            5
   × 0011          × 3
     0101           15
    0101
   0000
  0000
  0001111
```

Two's Complement

- Convention for handling signed numbers in binary representation
  - The leading bit is a sign bit
    - Binary number with leading 0 is positive
    - Binary number with leading 1 is negative
- Magnitude of positive numbers is just the binary representation
- Magnitude of negative numbers is found by performing the two's complement
  - Complement the bits
    - Replace all the 1's with 0's, and all the 0's with 1's
  - Add one to the complemented number
- Carry in most significant bit position is thrown away when performing arithmetic

Ch 1 / Foil 9

Two's Complement Example

- Performing two's complement on the decimal 7 to get -7
  - Using a five-bit representation

```
7 = 00111   Convert to binary

    11000   Complement the bits

    11000   Add 1 to the complement
  + 00001
    11001   Is  -7 in two's complement
```

Ch 1 / Foil 10

Two's Complement Arithmetic

- Computing 8 - 7 using a two's complement representation with five-bit numbers

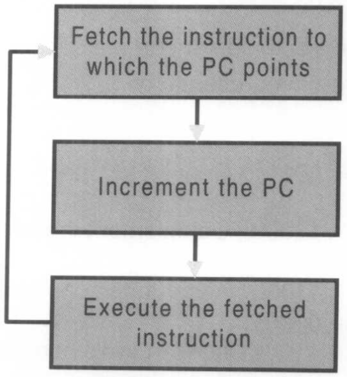
$$8 - 7 = 8 + (-7) = 1$$

```
                              01000   Two's complement of 8

                              11001   Two's complement of -7

     Throw away the           01000   Add 8 and -7
       high-order           + 11001
     carry as we are         100001
     using a five bit
     representation

                              00001   Is the five-bit result
```

Ch 1 / Foil 11

Control Unit

- The fetch/execute cycle is the steps the CPU takes to execute an instruction
- Performing the action specified by an instruction is known as "executing the instruction"
- The program counter (PC) holds the memory address of the next instruction


Fetch the instruction to which the PC points
Increment the PC
Execute the fetched instruction

Ch 1 / Foil 12

Input and Output Devices

- Accessories that allow computer to perform specific tasks
  - Receiving information for processing
  - Return the results of processing
  - Store information
- Common input and output devices
  - Speakers Mouse Scanner
  - Printer Joystick CD-ROM
  - Keyboard Microphone
- Some devices are capable of both input and output
  - Floppy drive
  - Hard drive
  - Magnetic tape units

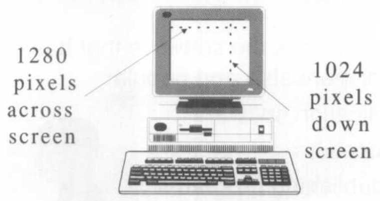
Ch 1 / Foil 13

Monitor

- Display device
- Also known as CRT (cathode ray tube)
- Operates like a television
- Controlled by an output device called a "graphics card"

Ch 1 / Foil 14

Monitor and Card Characteristics



- Refresh rate
  - How fast image is updated on the screen
- Resolution
  - Displayable area
    - Measured in dots per inch, dots are often referred to as pixels (short for picture element)
  - Standard resolution is 640 by 480
  - Some cards support resolution up to 1280 by 1024
- Number of colors supported

Ch 1 / Foil 15

Software

- Application software
  - ■ Programs designed to perform specific tasks that are transparent to the user
- System software
  - ■ Programs that support the execution and development of other programs
  - ■ Two major types
    - – Operating systems
    - – Translation systems

Ch 1 / Foil 16

Application Software

- Application software is the software that has made using computers indispensable and popular
- Common application software
  - ■ Word processors
  - ■ Desktop publishing programs
  - ■ Spreadsheets
  - ■ Presentation managers
  - ■ Drawing programs