A LABORATORY COURSE IN C++

NELL DALE

A Laboratory Course

in C++

Nell Dale
University of Texas, Austin

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE

World Headquarters
Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
www.jbpub.com

Jones and Bartlett Publishers Canada
P.O. Box 19020
Toronto, ON M5S 1X1
CANADA

Jones and Bartlett Publishers International
Barb House, Barb Mews
London W6 7PA
UK

Library of Congress Cataloging-in-Publication Data

Dale, Nell.
A laboratory course in C++ / Nell Dale.
p. cm.
ISBN 0-7637-0247-1
1. C++ (Computer program language) I. Title.
QA76.73.C153D336 1996
005.13'3--dc20 96-32748
CIP

Acquisitions Editor: David Geggis
Developmental Editor: Karen Jolie
Production Editor: Marilyn E. Rash
Manufacturing Manager: Dana L. Cerrito
Cover Design: Hannus Design Associates
Cover Printing: John P. Pow Company
Printing and Binding: Edwards Brothers, Inc.

Printed in the United States of America

00 99 98 10 9 8 7 6

Preface

Need for Support in Learning C++

For the last 16 years, the introductory computer science course has been taught mostly in Pascal. Over the last two to three years there has been a move toward C++ in place of Pascal. Even the strongest advocates for the change realize that C++ is going to be more difficult than Pascal for most beginning students to learn.

C and later C++ were designed for systems programming. Systems programmers are assumed to know what they mean and mean what they say. Therefore, C++ has very little runtime error checking and compiles some very weird code. Beginning students, on the other hand, often do not know what they mean and even more often do not mean what they say. Hence, it is essential that students understand the syntax and semantics of each construct as they go along. Closed laboratory activities seem an ideal way to make this happen.

Closed Laboratories in Computer Science

The Denning Report[1] introduced the term *closed laboratories* without defining exactly what they were. At least four different definitions subsequently surfaced.

1. A scheduled time when students work on their programming assignments under supervision.
2. A scheduled drill and practice time when students work on mini-problems under supervision.
3. The use of specially prepared laboratory materials where students interact with the computer as they would a microscope or Bunsen burner. The labs should help the student discover principles and solutions under supervision. This definition is closest to the spirit of the Denning Report.
4. A combination of two or more of the above.

With the publication of the Curriculum '91[2] report, laboratory exercises were suggested for many of the knowledge units. However, a precise definition of what constituted a closed laboratory activity was not included. And, in fact, many of the activities suggested could be done equally well in a nonsupervised (or open) setting.

Laboratory activities as defined in this manual are a combination of definitions 2 and 3.

---

[1]Denning, P. J. (chair) "Computing as a Discipline." *Communications of the ACM*, Vol. 32, No. 1, pp. 9-23.

[2]Tucker, A. B. (Ed.) "Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force." Final Draft, December 17, 1991. ACM Order Number 201910. IEEE Computer Society Press Order Number 2220.

Open versus Closed Laboratories

Although the Denning Report and Curriculum '91 imply that laboratory exercises should be done under supervision, we do not feel that this is essential. Our view is that closed laboratory exercises are valuable for two reasons: the exercises themselves and the extra contact time with a faculty member or a teaching assistant. If a closed laboratory environment is not an option, the students can still benefit from working the exercises on their own.

Organization of the Manual

Each chapter contains three types of activities: Prelab, Inlab, and Postlab. The Prelab activities include a reading review assignment and simple paper and pencil exercises. The Inlab activities are broken into lessons, each of which represents a concept covered in the chapter. Each lesson is broken into exercises that thoroughly demonstrate the concept. The Postlab exercises are a collection of outside programming assignments appropriate for each chapter. Each exercise requires that the students apply the concepts covered in the chapter.

When this manual is being used in a closed-laboratory setting, we suggest that the Prelab activities be done before the students come to lab. The students can spend the first few minutes of the laboratory checking their answers (Lesson 1 for each chapter). The Inlab activities are designed to take approximately two hours, the usual time for a closed laboratory. However, an instructor can tailor the chapter to the level of the class by only assigning a partial set of exercises or by shortening the time allowed.

The Postlab activities present a selection of programming projects. We do not suggest that all of them be assigned. In most cases, one should be sufficient, unless there are several related problems.

If the manual is not being used in a closed-laboratory setting, an instructor can assign all or a selection of the Inlab activities to be done independently (see the section "Flexibility" below). In either a closed or open setting, many of the Inlab and Postlab activities can be done in groups.

Theoretical Basis for the Activities

The decision to break each chapter in three types of activities is based on the work of Benjamin Bloom, who developed a taxonomy of six increasingly difficult levels of achievement in the cognitive domain.[3] In developing the activities for this manual, we combined Bloom's six categories into three. These categories are defined below in terms of the concrete example of learning an algorithm (or language-related construct).

Recognition The student can trace the algorithm and determine what the output should be for a given data set (no transfer).

Generation The student can generate a very similar algorithm (near transfer).

[3]Bloom, Benjamin *Taxonomy of Educational Objectives—Handbook I: Cognitive Domain.* New York: David McKay, 1956.

Projection The student can modify the algorithm to accomplish a major change (far transfer), can apply the algorithm in a different context, can combine related algorithms, and can compare algorithms.

The Prelab activities are at the recognition level. Most of the Inlab activities are at the generation level with a few projection-level activities included where appropriate. The Postlab activities are projection-level activities.

The activities are also influenced by the work of Kolb and others on how students learn.[4] The more actively involved students are in the learning process, the more they learn. Reading and writing are forms of active involvement. Therefore, the Prelab activities begin with a reading review, and many of the exercises ask the students to write explanations of what happened. Just watching a program run and looking at the answer is a passive activity, but having to write the answer down transforms the exercise into an active one.

Flexibility

A Laboratory Course in C++ is designed to allow the instructor maximum flexibility. Each chapter has an assignment cover sheet that provides a checklist in tabular form. The first column of the table in the Assignment Cover Sheet lists the chapter activities, in the second column students check which activities have been assigned, in the third column they record what output is to be turned in, and the fourth column is for the instructor to use for grading. The pages are perforated so that students can easily tear out sheets to turn in.

Student Disk

The accompanying disk contains the programs, program shells (partial programs), and data files. A copy of most of the programs or program shells is listed before the exercises that use the program or program shell. Programs used for debugging exercises are not shown, however. Because some of the exercises ask the student to go back to the original version a previous program or program shell, we suggest that the student copy the disk and work from the copy.

The disk is divided into subdirectories, one for each chapter. The programs and program shells are stored in files under the program name with a **`.cpp`** extension. Header files are stored with a **`.h`** extension.

---

[4]Svinicki, Marilla D., and Dixon Nancy M. "The Kolb Model Modified for Classroom Activities." *College Teaching*, Vol. 35, No. 4: Fall, pp. 141-146.

Acknowledgments

No author writes in a vacuum. There is always formal and informal feedback from colleagues. Thanks to those of you in my department who patiently answered "by the way" questions about C++. Thanks also to the following colleagues who wrote formal reviews of the manuscript: Mary D. Medley, Augusta College; Susan Wallace, University of North Florida; Paul Ross, Millersville University of Pennsylvania; Jeanine Ingber, University of New Mexico, Albequerque; James C. Miller, Bradley University; Ed Korntved, Northwest Nazarene College; Charles Dierbach, Towson State University; Mansar Zand, and University of Nebraska, Omaha. My special thanks to Mark Headington, University of Wisconsin, La Crosse, who must be the world's most meticulous reviewer and proofreader.

I want to add a special word of thanks to Porter Scobey at Saint Mary's University who discovered that there was a problem with recognizing the end of line under certain systems and then helped us solve the problem.

To Karen Jolie, a long-time colleague who always has the right answers; to Dianne Cannon Wood, who did a masterful job of copyediting; to Marilyn Rash, the production editor; and to all the staff at Jones and Bartlett: thank you. I trust that this is the beginning of a long and rewarding association.

Contents

1

Overview of Programming and Problem Solving

OBJECTIVES

- To be able to log on to a computer.
- To be able to do the following tasks on a computer.
  - Change the active (work) directory.
  - List the files in a directory.
- To be able to do the following tasks using an editor and a C++ compiler.
  - Load a file containing a program.
  - Alter a file containing a program.
  - Save a file.
  - Compile a program.
  - Run a program.
  - Change a program and rerun it.
  - Correct a program with errors.
  - Enter and run a program.
  - Exit the system.

Chapter 1: Assignment Cover Sheet

Name ______________________________ **Date** ______________________________

Section ______________________________

Fill in the following table showing which exercises have been assigned for each lesson and check what you are to submit: (1) lab sheets, (2) listings of output files, and/or (3) listings of programs. Your instructor or teaching assistant (TA) can use the Completed column for grading purposes.

Activities	**Assigned:** Check or list exercise numbers	**Submit** (1)	(2)	(3)	**Completed**
Prelab					
Review					
Prelab Assignment					
Inlab					
Lesson 1-1: Check Prelab Exercises					
Lesson 1-2: Basic File Operations					
Lesson 1-3: Compiling and Running a Program					
Lesson 1-4: Editing, Running, and Printing a Program File					
Lesson 1-5: Running a Program with an Error					
Lesson 1-6: Entering, Compiling, and Running a New Program					
Postlab					

Prelab Activities

Review

A computer is a programmable electronic device that can store, retrieve, and process data. The verbs *store*, *retrieve*, and *process* relate to the five basic physical components of the computer: the memory unit, the arithmetic/logic unit, the control unit, input devices, and output devices. These physical components are called computer *hardware*. The programs that are available to run on a computer are called *software*. Writing the programs that make up the software is called *programming*.

Programming

A program is a sequence of instructions written to perform a specific task. Programming is the process of defining the sequence of instructions. There are two phases in this process: determining the task that needs doing and expressing the solution in a sequence of instructions.

The process of programming always begins with a problem. Programs are not written in isolation; they are written to solve problems. Determining what needs to be done means outlining the solution to the problem. This first phase, then, is the problem-solving phase.

The second phase, expressing the solution in a sequence of instructions, is the implementation phase. Here, the general solution outlined in the problem-solving phase is converted into a specific solution (a program in a specific language). Testing is part of both phases. The general solution must be shown to be correct before it is translated into a program.

Let's demonstrate the process with the following problem.

Problem: Calculate the average rainfall over a period of days.

Discussion: To do the job "by hand," you would write down the number of inches of rain that had fallen each day. Then you would add the figures up and divide the total by the number of days. This is exactly the algorithm we use in the program.

Algorithm (on next page):