北京大学计算机科学技术系教材(影印本)

A LABORATORY COURSE IN C++

实验课程

NELL DALE

北京大学出版社

北京大学计算机科学技术系教材(影印版)

A LABORATORY COURSE IN C++

C++ 实 验 课 程

〔美〕 Nell Dale

北京大学出版社
北 京

图书在版编目(CIP)数据

C++实验课程：英文/(美)戴乐(Dale,N.)著. —北京：北京大学出版社，1999.1
ISBN 7-301-04018-0

Ⅰ.C… Ⅱ.戴… Ⅲ.C语言-程序设计-教材-英文
Ⅳ.TP312

书　　名：C++实验课
著作责任者：〔美〕内尔·戴乐
责任编辑：沈承凤
标准书号：ISBN 7-301-04018-0/TP·439
出 版 者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
网　　址：http://cbs.pku.edu.cn/cbs.htm
电　　话：出版部 62752015　发行部 62754140　编辑室 62752037
电子信箱：zpup@pup.pku.edu.cn
排 版 者：兴盛达激光照排中心
印 刷 者：国防科工委印刷厂印刷
发 行 者：北京大学出版社
经 销 者：新华书店
　　787毫米×1092毫米　16开本　15.625印张　554千字
　　1999年1月第一版　1999年1月第一次印刷
定　　价：19.00元

著作权合同登记　图字：01-1998-3011

出版前言

关于高等院校计算机专业程序设计入门课选择何种语言，观点不一，思想活跃，归纳起来有两大主流派：Pascal 派和 C++派。

Pascal 派认为：编程入门课不仅要让学生掌握一门具体语言的编程技术，更重要的是让学生树立良好的结构化程序设计思考方法和习惯，从这个意义上说，自然要首选 Pascal。

C++派认为：当今软件开发采用面向对象技术已是大势所趋，社会上已涌现出大量用面向对象的 C++语言开发的实用软件产品。如果学生在校正规地学习了一种程序语言，而在实际应用中面对的却是另一种不同的程序语言，为适应实际应用还需要进行可能是艰难的思维方式的转变，这种培养人的方法是低效的，不可取的。

Pascal 派认为：Pascal 并没有固步自封、墨守陈规，而是一直在紧跟程序设计技术发展潮流——Pascal 融入面向对象技术形成了 Delphi，这已是不争的事实。另外，无论从哪一种语言入门都会面临对其他程序语言的适应问题，这算不上什么困难，因为学习和掌握程序语言本来就应举一反三、触类旁通嘛！

C++派认为：正如已不必依照电子管—晶体管—小规模 IC—中大规模 IC—超大规模 IC 的次序学习计算机硬件一样，今天学习程序设计也应直接以面向对象技术的 C++为切入点，这在信息技术突飞猛进的今天，为增强技术竞争、人材竞争和市场竞争的实力来说是十分必要的。

Pascal 派认为：……。

C++派认为：……。

看！就像大家常见的电视辩论大奖赛一样，正方、反方都理由十足，谁也说服不了谁，最后评判输赢的根据不是各自所述论据，更不是论点，而是双方辩论的技巧。好了，既然如此，还是按照一位伟人所说的办：暂不要争论，先干起来，毕竟实践是检验真理的唯一标准。所以我们在积极推出《从标准 Pascal 到 Delphi 4.0》教材的同时，也积极支持用 C++语言作为编程入门语言。

然而令人苦恼的是，现已出版的 C++书籍极为缺乏配套的 C++程序练习题。机遇偏爱有心人。正当我们策划准备配套的 C++练习教材的时候，在今年北京举办的世界图书博览会上，我们看到了由美国德克萨斯大学奈尔戴乐等人编写，由琼奈斯·芭特雷出版公司出版的《A LABORATORY COURSE IN C++》一书，觉得它对于学生有针对性地练习并掌握 C++语言定会大有帮助。考虑到大学本科生阅读本书不会有什么困难，也为了师生尽早拿到合用的 C++练习教材，北京大学出版社经琼奈斯·芭特雷出版社同意，决定出版该书的影印本。

本书共分19章。各章依次为：(1) 程序设计和问题解决概述；(2) C++语法和语义以及程序开发步骤；(3) 算法表达式、函数调用和输出；(4) 程序设计和软件设计步骤；(5) 条件、逻辑表达式和选择控制结构；(6) 循环；(7) 函数；(8) 关于函数的作用域、生存时间及其他；(9) 附加控制结构；(10) 简单数据类型：内置类型和用户自定义类型；(11) 一维数组；(12) 应用数组：表和串；(13) 多维数组；(14) 记录(C++结构)；(15) 类和数据抽象；(16) 面向对象的软件开发；(17) 指针、动态数据和参照类型；(18) 链接结构；(19) 递归。本书内容丰富，习题精炼而有代表性，既可独立使用，也可作为相应教材的配套实验课程教材。每章都由三个环节构成：课前预备(Prelab)、实验课(Inlab)和课后练习

(Postlab)。课前预备指实验课之前应该预习、复习或阅读的内容，包括一些笔头作业；实验课主要具有针对性地练习，以便学生准确而全面深刻地掌握概念，一般都要求学生完成某个或某些相对独立的程序片断的编制；课后练习则要求学生设计一个完整的程序，以促进各章内容的融合贯通。

希望本书的引入会给大学程序设计入门课及相关课程的教学带来方便和效率，为教学的改革带来有益启示，也希望读者能把自己的体会、经验和意见及时反馈给我们，以便我们为您们提供更符合需要、更优良的服务。

本书中全部的源程序有配套软盘，需要的读者请与北京大学出版社理科编辑部联系购买。

北京大学计算机科学技术系　汪　琼
北　京　大　学　出　版　社　沈承凤
1998年12月26日

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 Report① 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 '9② 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 non-supervised (or open) setting.

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

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: reading material that includes

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. ACM Order Number 201910. IEEE Computer Society Press Order Number 2220.

a discussion of the topic, laboratory activities, and suggested programs to be done from scratch. The reading material includes paper and pencil self-help exercises, the answers to which are included in the appendix. The laboratory 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 suggested programs 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 laboratory 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. ① In developing the activities for this manual, we combined Bloom 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).

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. ② 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.

---

① Bloom, Benjamin *Taxonomy of Educational Objectives—Handbook I: Cognitive Domain*. New York: David McKay, 1956.

② Svinicki, Marilla D., and Dixon Nancy M. "The Kolb Model Modified for Classroom Activities" *College Teaching*, Vol. 35, No. 4, Fall, pp. 141—146.

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.

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):

Average Rainfall

```
Get total inches of rain
if total is zero then
   There wasn't any rain
otherwise
   Average <— total/number of days
```

Selection construct.
On thing is done if there is no rain; another thing is done if there is rain.

Function construct.
The task of getting the total number of inches is stated in the main algorithm without saying how it is done. This task is expanded separately as a subalgorithm.

Get total Inches of Rain

```
ofr number of days
      Get inches
      Add to total
```

Loop construct.
The task of getting the number of inches and adding it to the total is repeated for each day.

Sequence construct.
The task of getting inches is followed immeditately by the task of adding the inches to the total.

The C++ program that implements this algorithm is given below. Don't worry if you don't understand it. At this stage, you're not expected to. Before long, you will be able to understand all of it.

Incidentally, the information between the symbols /* and the symbols */ is meant for the human reader of the program. This kind of information is called a *comment* and is ignored by the C++ compiler. Information from // to

```
the end of a line is also a comment.

// Program Rain calculates the average rainfall over a period
// of days.  The number of days and the rain statistics are in // file Rain.in.

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

void GetInches(ifstream&, int&, float&);
// Rain statistics are read from a file; the average is
// returned.

int main()
{
    float average;          // average rainfall
    float totalRain;        // total accumulated rain
    int numberOfDays;       // number of days in calculation
    ifstream rainFile;      // data file
    cout.setf(ios::fixed, ios::floatfield);
    cout.setf(ios::showpoint);
    rainFile.open("Rain.In");
    GetInches(rainFile, numberOfDays, totalRain);
    if (totalRain == 0.0)
        cout << "There was no rain during this period."
             << endl;
    else
    {
        average = totalRain / numberOfDays;
        cout << "The average rain fall over "
             << numberOfDays;
        cout << " days is " << setw(1) << setprecision(3)
             << average << endl;
    }
    return 0;
}

//*************************************************
void GetInches(ifstream& rainFile, int& numberOfDays,
     float& totalRain)
// Pre:  rainFile has been opened.
//       numberOfDays is the first value on rainFile, followed
//       by numberOfDays real values representing inches of
//       rain.
// Post: numberOfDays is read from rainFile.
//       Number of inches of rain for numberOfDays are read
//       and their sum is returned in totalRain.
{
    float inches;       // Day's worth of rain
    int counter;        // loop control variable
    rainFile >> numberOfDays;
    totalRain = 0.0;
    counter = 1;
    while (counter <= numberOfDays)
    {
        rainFile >> inches;
        totalRain = totalRain + inches;
        counter++;
    }
}
```

Getting Started

Certain C++ systems provide an integrated environment for the creation and execution of C++ programs. For example, Turbo C++, which runs on IBM-compatible PCs, provides such an environment. The compiler, the editor, and the runtime system are all bundled together into one system. Other C++ compilers come separately so that you must use a general-purpose editor to enter your program. You compile the program and then run it. There are far too many systems for us to describe any of them in detail. Therefore, we use an analogy to describe the process of entering and running your program.

When you first get on a machine (*log in*), the *operating system* is the software that is running. You can think of the operating system as a hallway that connects all the other pieces of software. You enter the name of the software you want to use, and the operating system provides it. When you finish using the software, you must come back to the operating system (the hallway) before you can use another piece of software.

Each piece of software is like a doorway. The operating system opens the door and ushers you into the room where the software you want to use is kept. In the *editor*, you create a file of written information. This file may contain a program or data for a program.

An editor is a program that allows you to use your keyboard and screen like a very smart electronic typewriter. A file is the information that you type in through the keyboard. You see what you type on the screen. Commands to the editor do what you would do "by hand" with a typewriter. These allow you to change and rearrange letters, words, and sentences. A file resides in an area in secondary storage that has a name and is used to hold a collection of data. The data itself also is referred to as a file.

When you are satisfied with what you've typed, you give your file a name and tell the editor to save it for you. Giving a file a name is like putting information into a folder with a label on it. You can pick up the file and carry it with you from one room to another.

When you leave the editor, the operating system comes back with a prompt that says it's ready for you to tell it where you want to go next. If you've created a C++ program, you say that you want to *compile* (translate) your file (go to the C++ room). The operating system opens the door, and you hand your file folder to the C++ compiler.

The C++ compiler attempts to translate your program. If the program contains grammatical errors (errors in syntax), the compiler tells you so. You then have to go back to the editing room to correct the mistakes. When your C++ program finally compiles correctly, the C++ compiler leaves the object program (the translated code) in a file.

You tell the operating system that you are ready to run the object program. The file containing the translated program is taken into the execute room, where it is run. It's here, in the execute room, that the problem your program was written to solve actually gets solved.

When you're finished and ready to quit (*log out*), you tell the operating system. It opens the door marked exit, and you can leave.

Some systems bundle the editing room, the compiling room, and the executing room into one luxurious suite. For example, in Turbo C++ you enter this suite by saying "TC" to the operating system. You are then handed a menu of the things that you can do: create a new file, modify an old file, compile a program, or run a program. You even have the option of asking for help. You make a choice either by using a mouse (a pointing device) or by pressing a function key on the keyboard. Several of these choices then provide you with a second menu from which to choose. Integrated systems are *user-friendly* systems: you are never left stranded; you are always guided through the input (edit), compile, and run cycle by the use of these menus that appear at the top of your screen.

Chapter 1: Prelab Assignment

Name _____ **Date** _____

Section _____

Your instructor should provide a handout describing the system that you are using.

Exercise 1: What computer are you using?

Exercise 2: What operating system are you using?

Exercise 3: What C++ system are you using?

Exercise 4: Is your C++ system an integrated system, or is the editor separate from the compiler?

Exercise 5: If your editor is separate, which editor are you using?

Lesson 1-1: Check Prelab Exercises

Name _____ **Date** _____

Section _____

Exercise 1: What computer are you using? IBM-compatible PCs, Macintosh computers, and Digital Equipment computers are common in this course. You also might be using a terminal to a computer network.

Exercise 2: What operating system are you using? Windows '95, DOS, Macintosh, and UNIX are common operating systems.

Exercise 3: What C++ system are you using? There are lots of choices, but it must be compatible with the operating system that you are using.

Exercise 4: Is your C++ system an integrated system, or is the editor separate from the compiler? You only have two choices here: either it is integrated or it is not.

Exercise 5: If your editor is separate, which editor are you using? Word and WordPerfect are two common editors on PCs. Emacs and vi are two editors for UNIX machines.