A SET OF CS1 LABS UTILIZING GRAPHICAL OBJECTS AND INHERITANCE

Peter Andrews, Duane Broline, William Slough, Nancy Van Cleave[1]

Abstract *— Thirteen weekly CS1 labs utilizing C++ have been developed, implemented, and tested in a Linux student lab environment. Of particular interest is the emphasis on mathematics and the early and extensive use of objects and graphics, including simple animation. Many basic concepts (e.g., objects, files, and functions) are introduced and expanded over several weeks, and important themes recur throughout a series of labs (e.g., file processing and bar charts, classes and inheritance). Included are preview, lab, and follow-up exercises, check-off and grade sheets, weekly topics and hints for instructors, additional programming exercises that are extensions of the lab material, and a student reference manual. The labs are available online at www.eiu.edu/~mathcs/mat2170/index.html. They are intended for a lab session that runs approximately 100 minutes weekly.*

Index Terms — C++, CS1 Laboratories, Graphics, Objects

INTRODUCTION

Our CS1 course is required by math, math education, computer science, and pre-engineering majors. Hence, while striving to uphold high standards to facilitate transfer of the engineering students, we must also find ways to present the material so this wide spectrum of students can successfully complete the course. To this end, we have developed our own series of computer laboratories that are challenging and thorough, and which directly engage our students in the learning process.

It is difficult to find CS1 C++ based laboratories which stimulate student critical thinking skills, are well integrated both with the course and as a series, and which provide a variety of activities appealing to a broad range of student interests while maintaining a high standard of rigor. We have designed, implemented, and tested in a Linux environment, thirteen CS1 labs. Each lab provides pre-lab and in-lab exercises as well as follow-up programming assignments. Our lab sessions are scheduled for 100 minutes, one day per week. The majority of students can complete most of the labs in that time period. Other support materials include references to corresponding reading in Cohoon and Davidson's *C++ Program Design* textbook [1], lab check-off sheets, lists of topics and hints for the instructor, and gradesheets to accompany each lab. The textbook does have a lab manual [2] to accompany it, but we chose to create our own labs in order to incorporate more graphics and tailor the exercises to our preferences.

Of particular interest with respect to these labs is the extensive use of graphics (introduced by Week 4), including simple animation (Week 6), that helps capture student attention and imagination in order to improve the learning experience. We have utilized and extended Cohoon and Davidson's EzWindows graphics library, available through their website: *www.mhhe.com/c++programdesign*, which provides such graphics primitives as RectangleShape, CircleShape, and Label classes. Animation is accomplished by erasing, repositioning, and redrawing shapes. This simplicity is what enables students to do nontrivial graphics in their first semester of C++ programming. We have had students create programs which bounce a rectangle off the four sides of a window, calculate and trace trajectories on the screen, and display various Lissajous curves. Most students are impressed with the new abilities they are able to acquire in a relatively short time. This simple graphics package allows us to make assignments both accessible to and exciting for the average student while still engaging the better students and providing them with easy avenues for extending themselves. For example, one pair of students who introduced their Lissajous program with a flashing French flag

LAB SUBDIVISIONS

Each lab consists of nine parts – five for students, three for the instructor, plus a Makefile for the particular lab. Specifically, we provide:

- references to corresponding reading in textbook
- preview exercises
- preview submission sheet (a subset of preview exercises)
- lab exercises (including multiple programs and header files)
- follow-up programming exercises
- lab check-off sheet
- list of topics and hints for the instructor
- gradesheet for each lab
- makefile (unique to the particular lab)

Students are expected to complete the preview before lab, and are required to hand in the preview submission sheet

[1] Eastern IL University, Math Dept, 600 Lincoln Ave, Charleston, IL 61920, cfpga@eiu.edu, cfdmb@eiu.edu, cfwas@eiu.edu, cfnkv@eiu.edu

0-7803-6669-7/01/$10.00 © 2001 IEEE **October 10 - 13, 2001 Reno, NV**
31st ASEE/IEEE Frontiers in Education Conference

at the beginning of the lab, along with their solution to the follow-up programming assignment from the previous week's lab. This ensures students are familiar with the lab exercises and have had an opportunity to think about possible solutions before sitting down at a computer (or at least have been encouraged in this direction).

During lab, students work in groups of two or three. Over the course of the last few years we have tried several different administrative regimes. In some sections we have allowed groups to form early and remain relatively static, in others we have required students to change partners for each lab, affording them an opportunity to work at least once with most of the other students in the class. At times we have asked group members to work in parallel, each using their own computer although sitting next to their partner(s). We have also required each group to work collectively on a single computer, taking turns with the typing and checking then sharing files at the end of the lab. There does not seem to be a single best option among these choices. Each of these schemes has its strengths and weaknesses.

When all students in a group complete specially marked exercises, they must show and/or explain their solution to the instructor and obtain his or her signature on the Check-Off sheet. This gives the instructor a chance to visit with each group and to clarify any problems that may arise. It avoids, to some extent, the scenario where a handful of students monopolize the instructor, leaving the rest of the class to fend for themselves. It also is a deterrent to those students who might be tempted to skip exercises in order to finish lab as quickly as possible. The first few weeks of the semester we often help in each other's labs in order to prevent groups from becoming stuck too long on problems and to answer the plethora of inevitable questions beginners have.

Each week a follow-up programming exercise is included which ties together concepts both from previous weeks and from the current set of exercises. This reinforces basic ideas, and allows students the "ah ha!" experience of seeing the patterns in programming begin to emerge as they apply tools and concepts (they either have already learned or are in the process of acquiring) to a problem which is new but that contains elements they have just seen in lab or have seen in prior weeks.

For example, Week 5 combines the graphics objects learned the previous week with an introduction to loops. The first program in the exercises acquaints the students with a few common implementation and logic mistakes that cause infinite loops. The second program creates tables of values (for the summation of terms of a geometric sequence) and has students determine the closed formulae for these summations. The third program begins with five squares drawn on a diagonal from upper left to lower right. Students must increase the number of squares drawn to ten, modify the program so the squares decrease in size, then modify it once again so the squares are drawn until they disappear. The last lab exercise works with nested loops which create stacks of concentric circles in alternating colors along a diagonal. Students modify the program so the stacks are drawn first horizontally, then vertically. Once they've worked out the details of how to draw the nested circles vertically, they must then move each circle horizontally so as to touch its neighbors on the left and right in a diminishing line. The follow-up exercise is to create an 8×8 checkerboard with immobile playing pieces. As soon as the student realizes the relationship to the nested circles exercise, the coding is straightforward.

We have developed several additional programming assignments for individual projects that expand and enhance the lab exercises. Also provided is a student lab manual which contains informal descriptions of some of the classes that are used in the labs, both those provided in EzWindows and those that were developed by the authors. These include the necessary library name, constructor or initialization information, messages (with header information such as return type and type, order, and default value of parameters), operators and auxiliary functions (if any). For those developed locally, we also included the messages each derived class inherited from its parent class.

In addition, the student reference manual contains short descriptions of standard library functions such as those found in ctype, cmath, and iomanip, as well as a table of the ASCII Character Set collating sequence. The last pages give information on accessing the Linux lab server remotely using telnet, pico, and g++ to create, compile, and execute non-graphical programs.

THEMES

Whenever possible, throughout the series of labs, concepts introduced in one or more of the lab programming exercises of one week, are expanded upon the next week by involving some programming by the student. By the third week students are expected to do a significant portion of the programming associated with the concept. Thus, students have an opportunity to become familiar with new terminology, have seen the implementation of the concept, and ultimately have a reference source to consult when the time comes for them to do their own coding.

One of the difficulties of a first programming class is the terminology associated with an entire discipline with which most students are not familiar. As they simultaneously attempt to grasp both the new words and the foreign notions presented in lecture, and to understand the more comprehensive textbook descriptions, all the new terms and concepts can become jumbled together. By easing into the harder concepts, then reinforcing what students have learned by revisiting the topics in later labs, we enable students to understand and retain more than if they get a single assignment which covers the concept, then go on to the next

topic. Some of the more interesting themes are presented below.

INPUT FILES

Over a five week period, we present the concepts of input files, reading from files by either the extraction operator or the get message, obtaining file names from the user, and checking for errors on opening the file. File input is first seen by the students, in two Week 6 programs, as complete code not requiring any modification by them. File names are hardcoded into the program so the students quickly see the tediousness of the necessary editing and recompiling to change input file names. The loop that reads strings until the file is exhausted is also provided for them.

The next week, the code for setting up an input file is again part of what is given to them in the lab exercises, but we progress to obtaining the filename from the user and to using the get message in order to obtain a character at a time from the file. In Week 9's follow-up the students themselves must supply the code to get the filename and successfully open an input file. By this point, if they are at a loss, we can suggest they've seen this before and point them to previous labs to find the code. Finally in Week 10's follow-up, the students must provide their own code for continuing to prompt for and get a file name from the user until they obtain one corresponding to a file which can be opened successfully.

FUNCTIONS

The same technique is applied to functions. From Week 3 throughout the rest of the semester, students use predefined classes and send messages to objects. In Week 3, they are shown classes for a psuedo-random number generator, an ordered pair, and strings. The intent at this point is just to familiarize them with the concepts of classes and message passing to objects.

In Week 4 they're introduced to the graphics classes we'll be using for the rest of the semester, and practice sending messages to objects of these classes as well as to the classes they saw the previous week. The next few weeks, for the classes they already studied in the previous weeks, they learn about and practice using more messages in the context of learning about control structures. In Week 6 we include a user-defined function, sleep, which delays execution of the program to slow down the display of graphics. Students get to see its prototype, its definition, and how it is used in the exercise program.

In Week 7 we review what we have learned about functions and messages thus far in the semester, and introduce the ctype, cmath, and iomanip libraries. The students' previous exposure to messages, their reference manuals, and their experience with functions in mathematics help prepare them to understand and quickly utilize these library functions. After this week, students will incorporate these libraries repeatedly in their programs.

We begin utilizing header files in Week 8, giving students only the compiled version of the implementation file so they are guided to focus on the function interfaces. Here, we introduce the concept of testing, and have them determine which of several function provided in the lab exercises contain logic errors. This must be done through testing alone since they don't have access to the source code.

In Week 9 they write function definitions, for example, Max and Min with both two and three parameters. This is where the early investment in using messages pays off as they are already familiar with the concept of return types, parameter types, importance of parameter order, and default parameters from a user's point of view.

The next week, they explore reference and value parameters, as well as function templates. In Week 11 they implement member functions, now a fairly small step from using them and writing stand-alone functions, and then are expected to write functions for the remainder of the semester.

FILE PROCESSING / BAR CHARTS / SCALING

This is an interesting theme that allows students to see how solutions to previous, related problems can be utilized to solve new problems they encounter. Thus the practice of programming can be seen more clearly as a building process rather than as a series of unrelated programming exercises which focus on a single concept that appears to be disjoint from all other assignments.

For the Week 6 follow-up exercise, students must complete a skeleton program in order to obtain statistics about a textfile, such as counts of short, medium and long words, average word length, and the length of the longest word in the file. Once these statistics are obtained, they must then open a graphics window and display a bar chart with three bars, appropriately scaled and labeled, displaying the percentage of words in each category (short, medium, and long).

Week 7's follow-up requires students to scale and translate a Lissajous curve from abstract Cartesian coordinates to a graphics window.

Processing a file and displaying information graphically is again seen in Week 10 when the follow-up requires students to make two passes through an input file of positive integers. The first pass finds the largest value and counts the entries; the second displays a bar in a chart representing each number scaled so the largest has height equal to the height of the window. Having done a simpler bar chart before and encountering scaling for the third time allows students to tackle this program with some degree of confidence.

In Week 12, students approximate the area under the graph of a non-negative function in a manner similar to a Riemann Sum. They must sum the areas of rectangles all having the same width, but with height determined by the

function, under a given curve. By this point in the semester they use vectors to store the heights of the rectangles and complete a function which displays the bars (rectangles) in a graphics window, again scaling them to fit.

INHERITANCE

Inheritance is approached in a similar fashion, albeit by necessity in a more accelerated form. Having worked with RectangleShapes since Week 4, we use them to demonstrate the power of inheritance to our students in Week 11. The first class we derive is FramedRectangleShape, which provides a frame around a RectangleShape by the addition of data members for frame width and color, and an additional rectangle which underlies the RectangleShape in order to provide the frame around it. Students can easily see the need for a new Draw() member function which utilizes the RectangleShape's Draw() to display the inner, framed rectangle. Since no Erase() function is provided, the students become involved in a discovery learning experience when they see the results of using the inherited Erase(), and are pleased to find they are able to write the necessary member function for the derived class.

This rather gentle introduction to inheritance is extended by deriving two more classes. From FramedRectangleShape we derive the Board class, which adds blocks that alternate in color, i.e., a checkerboard with a border (frame) around it. The additional information associated with this class is the number of blocks on one side of the board (so we could play checkers or tic-tac-toe), and the two colors used for the blocks.

From Board, we derive GameBoard, along with the associated class Piece, used to represent a marker or checker. Along the way, students are required to complete various member functions, but one draw back of our labs is that they do not develop any class from the design stage through to implementation.

The last evolutionary step in this inheritance series from RectangleShape is a FractalBoard, also derived from the Board class. Previously, students had completed the Board member function GetBlockColor(), which alternated two colors (traditionally red and black, but potentially any of the eight colors provided in EzWindows) based on the block's position (row and column) in the board. For the new FractalBoard class, students rewrite GetBlockColor() so that it calculates the Julia number of the block, based on its row and column, and returns the associated color (one of eight). Students are quite pleased with themselves when they are able to produce fractals similar to samples they've seen in textbooks and on websites. It is also easy for them to see the effects of resolution on the smoothness of the curves, just as they did in the program to calculate the area under a curve.

GRAPHICS

Graphics and animation, even with a limited number of colors, is the sugar that makes the work of learning to program palatable for some students. Hence we've incorporated graphics in as many laboratories as possible. Hidden within this entertaining aspect of the course is the practice of learning to think logically and working with mathematical and the usual CS1 programming concepts.

We begin using simple graphics in Week 4, where students are initiated to the libraries and messages. They create and display a variety of shapes in a range of sizes, colors, and positions. Here, the students are rewarded for using messages correctly by the appearance of the requisite forms in the graphing window. Moreover, hand calculations are not necessary to reveal most errors. Figures either appear where they are supposed to or they don't!

The following week, concentric circles are used to allow students to observe the behavior of nested loops, and to practice modifying these loops to change the orientation of the output. Because of the alternating colors, and some (common) mistakes which produce interesting results (such as a strobing marquee), students are kept attentive while trying to solve the different exercises.

In Week 6 as we study selection statements, the students are led to the problem of bouncing a rectangle off the sides of the graphing window. Most are excited when they accomplish this, especially given they are only six weeks into their first programming course.

In the following weeks, students become adept at using graphics to represent information (see section on bar charts above). Although it is a challenge, the Week 7 follow-up exercise of displaying a Lissajous curve brings much satisfaction to students when they first see the curve appear on their screen, created by the program which they themselves wrote. We also make the individual programming projects (usually two per semester) graphics based as well. A few examples are the problems of displaying a tower of squares, displaying the trajectory of a banana given the angle of release and its initial velocity, and creating and displaying a fractal. Most students find these types of problems to be more interesting than, say, purely mathematical or business-oriented problems, yet they exercise the same skills.

CONCLUSION

This series of thirteen labs represents a cohesive and comprehensive approach to introducing students to programming in C++. Basic concepts and themes are used and reviewed throughout the series, reinforcing the most important objectives. In order to keep pace with the material, students must devote time and thought to the lab previews, which helps prepare them for and enriches their in-

lab experience. It quickly becomes obvious they cannot proceed without this preparation, so it puts the responsibility on their shoulders to spend the necessary time outside of lab working on these details.

Although the labs are not perfect, there are many facets to recommend them. They seem to strike a good balance between teaching basic programming skills, while incorporating the object-oriented aspects of C++. While students do not receive much practice in designing classes, they do understand the concept of member data and functions, are able to implement these given a design, and are able to utilize classes in their own programs.

The use of graphics to enhance student attentiveness and interest also seems to work well with these labs. For visual learners, it is a definite bonus to be able to actually see the results of their programs in pictorial form rather than as simply textual output.

Overall these labs form a solid basis for an introductory course in C++ programming for a wide range of students. It covers the usual basics from assignment statements and arithmetic operations to classes, objects, and member functions. If you are interested in obtaining files that are not online, please feel free to contact any of the authors.

REFERENCES

[1] Cohoon, James P., and Jack W. Davidson, *C++ Program Design: An Introduction to Programming and Object-Oriented Design*, McGraw-Hill, 1999.

[2] Cohoon, James P., and Jack W. Davidson, *Laboratory Manual with Lecture Notes for use with C++ Program Design: An Introduction to Programming and Object-Oriented Design*, McGraw-Hill, 1999.