LA BELLA FIGURA: MAKING A GOOD IMPRESSION WHEN TEACHING AN INTRODUCTION TO PROGRAMMING TO NON-ENGINEERS

Ani Nahapetian
Computer Science Department
University of Los Angeles, California (UCLA)
Los Angeles, CA 9005
ani@cs.ucla.edu

ABSTRACT

This paper presents real and practical solutions to teaching an introductory course in programming to non-engineering students. It addresses the challenges and the potentials of making a good first impression in such a course, including the desire to fit everything into one quarter, the opportunity to encourage students to explore programming further, and the reality that students are looking for a useful tool for use in their respective fields. Specific suggestions and ideas for course content presentation, textbook selection, and course projects are presented that address these challenges.

1. INTRODUCTION

An introduction to programming course for non-majors, and more specifically non-engineers, is a uniquely challenging task. It is composed of students from many different majors including those in the arts, humanities, sciences, and Mathematics. However, it can provide a great opportunity to make a good impression regarding the field of Computer Science and the utility of specific concepts.

There is generally a factor of intimidation and fear associated with such classes, because they are so distinctly unique from all other course offerings in the arts, humanities, and sciences, and because they require a large amount of project work outside of the classroom. Additionally, students are not always clear about how they can apply the knowledge learned from a programming class to their individual fields of study. [6] The course can be an excellent opportunity to correct the misperceptions that exist about the field of informatics generally, and computer programming specifically. Also, if instructors intentionally focus the student's attention to programming as a tool instead of as a combination of constructs and tricky syntax, through specific concepts or application areas, students can improve their opinion of programming in general.

Course instructors are often faced with the challenge that most students only take the first course in the three programming course series. This differentiation from traditional programming courses targeted to Computer Science students, who are required to take all the courses in the programming series, creates the challenges of determining an effective and appropriate course and project schedule, while at the same time enticing students to further explore the field of Computer Science and the power of programming.

This paper is based on the experience of developing and of teaching of an introductory course for non-engineers at the University of California, Los Angeles. Contrary to the introduction to programming course, CS 31, for Computer Science engineering majors, the PIC 10A course targets the general student body. It has three main contributions, which are based on actual classroom experiences and a survey of the related work. First, the basis for the approach to such a

class is presented, including university conventions. General considerations for course design are provided. Motivated by the underlying themes presented, a series of topics that this paper proposes is critical to include in an introductory programming class, specifically using C++, for non-engineering students is outlined.

2. COURSE INFRASTRUCTURE

The University of California, Los Angeles (UCLA) is large research focused public university in California. It has an undergraduate population of nearly 27,000 students, where over 11% of the students are enrolled in the engineering school. The programming course offerings at UCLA are split between the Computer Science department and the Mathematics department's Program in Computing. This resulting split is due to a historical decision to have only engineering school students enroll in the Computer Science department's courses. As a result, the Math department created the Program in Computing to serve the Informatics education needs of the rest of the university.

My experience teaching spans both introductory programming courses: the one for engineers, called CS 31, and the one for non-engineers, called PIC 10A. With the exception of a certain group of students, including Math and Linguistics majors, PIC 10A is not a required course. Students can take the course to fulfill certain logical reasoning general education requirements, however.

Introduction to programming is presented using C++. There exists another complementary course which is an introduction to programming in Java. However, the C++ course is the initial starting point to programming that is offered to non-engineering students. There are two other courses in the C++ programming series, called PIC 10B and PIC 10C, which cover more advanced topics, including inheritance, polymorphism, and advanced data structures.

3. BASIS FOR APPROACH

The introduction to programming course for non-engineers was a particularly interesting course to plan and prepare, due to its unique challenges and considerations. The following observations, trends, and conclusions colored my perspective in preparing and delivering the course content.

It is commonly believed among the student body that the Program in Computing classes are especially challenging. This perception is confirmed by the fact that they are weighted 25% more heavily than most other classes. A similar opinion exists in the engineering school with regards to the programming classes offered to the engineers. This common misconception is not unique to UCLA. [1] [9] Students often come to the class without prior knowledge of programming, which further intimidates them. This is dissimilar to the Computer Science course, where there are often Computer Science majors who taken and enjoyed a programming class in high school.

The intimidation factor has been shown to be especially detrimental to women. It has been shown that students who may have a strong aptitude from the subject but lack experience and confidence do not explore the field further [1].

Computer Science courses targeted to non-majors in general, and to non-engineers in this case, provide an opportunity to change the negative perception of the field. Computer Science is often perceived as a dry field, where syntax plays a vital role. [9] An introductory course can educate the student population by demonstrating its application oriented nature, by highlighting the field's love of puzzles and games, and by showing how easily Computer Science supports ideas

and practices in other fields. Even the demonstration that computer scientists have a broad range of interests can be surprising and inspiring for students seeking a broader experience in college.

Introductory classes to non-engineers provide an ideal opportunity to present the power of programming as a tool for experts in other fields. In addition to demonstrating the applicability and utility of employing programming in various fields, a broad range of application areas can spark a strong interest and encourage a passionate response by students. For example, a typical programming exercise for practicing the concepts of strings and nested looping is the development of a subsequence matching program. A simple modification of the problem to a gene sequencing application to determine the percentage of relation between genes can help engage Biology students, as well as provide an opportunity to mention the growing field of Computational Biology and its influence on genomics.

More detailed examination of this point has been carried in various classes and publications. [2] [7] [8] [10] [13] [14]. In a related fashion, [3] [5] use programming projects to give a glimpse into the more advanced topics in Computer Science, instead of using applications from other fields.

One significant phenomenon with non-engineering students is that they take one course in programming and many do not move on to learn about the more advanced constructs of programming, data structures, or algorithms. This can be due to two reasons at UCLA: 1) only one programming course is required for several majors, including mathematics, linguistics, and statistics, and 2) students who take a programming course for enjoyment are still faced with strict unit and graduate requirements and generally can only spare one class from their tightly scheduled four year graduate plans. As a result, providing students with exposure to some critical concepts can enable them to search for further details during their programming practice and after they have completed the course.

4. PROVIDING AN OVERVIEW

The phenomenon where non-engineering students only have an opportunity to take one programming course influences topic choices and their presentation ordering. Providing students with exposure to some critical concepts encourages them to search for further details, after they have completed the course.

The presentation of object oriented programming (OOP) from the beginning of the course, I argue is important. OOP is the standard programming practice, yet its introduction is often withheld until them end of the course schedule. This many not be detrimental for Computer Science students who will go on to further explore OOP in their other programming courses. However, if only a single programming course is taken, the lack of exposure to OOP can prevent the level comfort necessary to use it at all or to deal with its use in legacy code.

In the case of the C++ programming language, the power of the standard template library is immeasurable. If students are not exposed to it at all, countless hours can be wasted coding what can simply be imported. The very different notation can seem overwhelming to students, who do not realize that the additional leap in thinking is small, once objects and classes are understood.

Discussing vector immediately after the presentation of arrays addresses this issue.Vectors are a more robust construct than arrays, and thus student adopt their use early. Once the concepts of OOP are made clearer, the leap to a template class is eased by the early exposure. Then the mere mention of other template classes strongly increases the chance of a student exploring the other templates and having an easier time applying them to future code.

Non-engineering students who program for research or for their job will almost definitely be working with text files. Thus, the presentation of file I/O in the introductory class is beneficial and useful. Additionally, the explanation of streams seems to answer many questions that students have in their minds about how computer systems deal with real-time input. Additionally, if file I/O is presented early in the quarter, it can allow a greater level of project sophistication.

There is a debate about when to present recursion. Many textbooks present recursion in conjunction with functions. However, Computer Science programming courses have been known to skip the topic in favor of presenting it with stacks in more advanced courses. Recursion is one of the most unique and interesting topics in programming. It provides for small code, which again is appealing to students. Finally, introducing students to a new way of thinking may just be the only part of the programming course that some students take with them, and so it would be a shame to not expose students to the challenging but fascinating concept of recursion.

5. CONCEPTS: ON WHAT TO FOCUS STUDENTS

The shift in focus from constructs to tools can be a powerful motivator for non-engineers to explore programming. One of the true advantages to learning how to program is its application to other fields. The following applications of programming presented here include specific mention of lecture examples and programming projects.

5.1 Simulation

One important use of computing in other fields is the extraction of computer simulation results. The presentation of simulation, which occurs in conjunction with the presentation of the pseudorandom generators, is joined with the concept of functions. The syntax of functions and even their huge power in enabling large scale program development, abstraction, or software engineering is distinct from the incredible power of computer simulation. I propose that the concepts of functions and simulation be separated, specifically, for non-majors, in an attempt to emphasize both. They are two distinct topics despite their overlap in almost all programming texts.

There exist a broad range of interesting examples, project assignments, and lab work that can be used to reinforce the concept of simulation, to demonstrate its potential benefits, and to show examples where simple code is hugely powerful. The following are some interesting examples.

1) Figuring out the value of pi by virtually throwing darts randomly at a circle embedded in a square is a well suited application to a class with a large number of Mathematics majors. A calculation of the ratio of darts in the circle versus the numbers of total darts gives a numerical value for pi. [12] Precision is enhanced by increasing the number of dart throws. This requires only simple programming skills, specifically knowledge of arithmetic operators and iteration.
2) Agent based simulation research bridges the fields of simulation and animal behavior, by defining rules that govern individual behavior and by then observing the resulting effects of the rules on societies through simulation. This research area has inspired a simulation project, which observes group behavior after individual actors have their behavior biased towards certain actions. One project involves determining how long civilizations will survive, if each individual kills, say, 50% of the people from a different country, assuming an interaction rate of once per year. This problem only requires knowledge of nested iteration and arrays. (The results are surprising for peace lovers.) Such simulation can be interesting to the significant number of social science students who take the course.
3) Revealing statistical phenomenon by using simulation, such as with the classic drunken walk problem [12], is interesting to statistics students as well as social students who take a significant amount of Statistics classes.

4) Simulation of the Monty Hall problem [11] (of whether to switch doors, after guessing which door reveals a prize) is an exciting way to mention some television history in a programming classroom.

5.2 Visualization

Visualization is an important component of various research fields, including Computational Chemistry, Statistics, and Public Health. The ability to visualize data can help develop theories in the social sciences and humanities and analyze data in the sciences. Enabling skills for visualization range from the most basic, such as numerical value display to the more sophisticated graphics package usage options. In an attempt to provide an impression of the area the following programs have been useful.

1) In the presentation of arrays, one of the first examples used involves the displays data in a rudimentary bar graph composed of asterisks. [4]
2) In lecture, the extraction of a conclusion from a large corpus of data encourages student involvement. The example involves questioning students for a conclusion, coding a simple display summary of the data, and then again attempting to draw a conclusion about the data. Data from a bell curve has been my choice in the past
3) Finally, the following project is a way to expose students to library functions. Instead of tedious examples from the math library, examples are done using OpenGL, the graphic library in C++. The graphical display is exciting for many students. One possible project is to have students reverse engineer the code for a digitally generated image.

5.3 Text and Image Analysis

The ubiquitous nature of the internet and the large corpus of text data on the World Wide Web have enabled the spread of text analysis and text mining approaches for various needs. The introduction to such techniques can have a very broad level of applicability in many fields, especially for students in linguistics, psychology, and sociology. The following are some text and image analysis problems of various levels of difficulty.

1) The Flesch readability index problem [11] is lengthy text analysis project, which rates the readability of text in a file using a formula related to word and sentence length. Students enjoy testing their own college papers for a readability score. It reinforces may of the concepts presented through out an introductory programming class.
2) A translation example which leverages statistical techniques is a great project for exciting linguistics students. It reveals a great deal about the way translation is accomplished today in pervasive tools such as Google translate, and thus it can be an illuminating experience for all majors, as it makes a tangible connection to the state-of-the-art in Computer Science research. It allows students to build confidence in their skills, to help students begin to consider the option of graduate school. The translation project involves students reading files that are direct translations of each other, making statistical inferences, and then passing through testing files for translation. This project requires knowledge of file I/O.
3) A simple silhouetting program that prints an asterisk if a pixel is greater than a certain value and a space otherwise can display impressive images to the screen. With the help of some template code that skips over file I/O in the first weeks of the course, when student only have the skills to print to the screen and to do some simple comparisons, the image analysis example appears as very substantial accomplishment.

6. CONCLUSION

In this paper, a practical approach to teaching an introduction to programming was presented. Specific issues related to teaching the course to non-engineering students is presented including, their intimidation factor, the large numbers of students who do not continue with the advanced programming courses, and the requirement to provide an intellectually challenging course in the programming language of C++. Various programming projects are presented, with an explanation of their appropriateness for such a course. It was argued that the material does not need to be modified, but instead the topic ordering and the choice programming projects should be tailored to the students' needs and interests.

7. REFERENCES

[1] Carter, L. 2006. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06.

[2] Cortina, T. J. 2007. An introduction to computer science for non-majors using principles of computation. SIGCSE Bull. 39, 1 (Mar. 2007), 218-222.

[3] Marie desJardins, Michael Littman, Broadening student enthusiasm for computer science with a great insights course. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, (Milwaukee, Wisconsin, USA, March 10-13, 2010).

[4] Deitel, H. M. and Deitel, P. J. 2002 *C++ how to Program, Fourth Edition*. 4th. Prentice Hall Professional Technical Reference.

[5] Dodds, Z., Libeskind-Hadas, R., Alvarado, C., Kuenning, G. 2008. Evaluating a breadth-first cs 1 for scientists. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, (Portland, OR, USA, March 12-15, 2008)

[6] Forte, A., Guzdial, M. (2005) "Motivation and Non-Majors in CS1: Identifying Discrete Audiences for Introductory Computer Science" *IEEE Transactions on Education*. 48(2), 248-253.

[7] Gimenez, O., Petit, J., and Roura, S. 2009. Programacio 1: A pure problem-oriented approach for a CS1 course. In *Procedings of the 4th Informatics Education Europe* (Freiburg, Germany, November 2009).

[8] Goldman, K. J. 2004. A concepts-first introduction to computer science. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004).

[9] Hatziapostolou, T., Sotiriadou, A., and Kefalas, P. Promoting Computer Science programmes to potential students: 10 Myths for Computer Science. In *Proceedings of the 3rd Informatics Education Europe* (Venice, Italy, December 2008).

[10] Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., and Hosking, A. L. 2009. A multidisciplinary approach towards computational thinking for science majors. *SIGCSE Bull.* 41, 1 (Mar. 2009), 183-187.

[11] Horstmann, C. and Budd, T. 2008 *Big C++*. 2nd. Wiley Publishing.

[12] R. Sedgewick and K. Wayne. Introduction to Programming in Java: An Interdisciplinary Approach. Addison Wesley, 2008.

[13] Smarkusky, D. L., Toman, S. A. 2009. An interdisciplinary approach in applying fundamental concepts. In *Proceedings of the 10th ACM conference on SIG-information technology education*, (Fairfax, Virginia, USA October 22-24, 2009).

[14] Zhang, M., Lundak, E., Lin, C., Gegg-Harrison, T., Francioni, J. 2007. Interdisciplinary application tracks in an undergraduate computer science curriculum. In *Proceedings of the 38th SIGCSE technical symposium on computer science education*, (Covington, Kentucky, USA, March 07-11, 2007).