A PROGRAMMING COURSE: TO FOSTER CRITICAL THINKING IN LIBERAL ARTS STUDENTS' MIND

Sarah Tasneem
Eastern Connecticut State University
Willimantic, CT 06226
tasneems@easternct.edu

ABSTRACT

An introductory programming course can instill critical thinking, logical reasoning and problem solving skills to liberal arts students which they can later apply to their respective disciplines. Computer Programming is the process of solving problems. However, learning programming is a complicated business as it involves multiple processes and various skills. A profound understanding of the design activity of successive decomposition in programming may result in significant educational benefits in many areas, including those unrelated to computer science.

This paper presents an approach to teach an introductory programming course focusing at first on the development of the steps of algorithm using pseudocode, instead on the syntax of the language itself. For beginners programming language syntax can be very discouraging and intimidating. Student's interests are withdrawn if they are introduced to the syntax at the very beginning. At this stage of learning the goal should be capturing the essence of designing a solution instead of focusing on the complexity of the programming language implementation. Laboratory experiments are designed to solve small scale real world problems where students acquire hands on experience through algorithm animation. Bringing an algorithm in life through animation can help students understand logistics what the algorithm does and how it works. Most importantly, the course aims to help students, irrespective of their major, feel confident to write small programs accomplishing useful goals.

1. INTRODUCTION AND BACKGROUND

MIT Computer Scientist Seymour Papert in his seminal book (Mindstorms: Children, Computers and Powerful Ideas) mentioned that a deep understanding of programming, in particular the design activity of successive decomposition as a mode of analysis, results in significant educational benefits in many domains of discourse, including those unrelated to computers and information technology [1]. A recent National Academy of Sciences report states that programming knowledge and experience is beneficial to everyone in an information society. "The continual use of abstract thinking in programming can guide and discipline one's approach to problems in a way that has value well beyond the information technology-programming setting. In essence, programming becomes a laboratory for discussing and developing valuable life skills, as well as one element of the foundation for learning about other subjects [2]". It is identified in the report that exposure to programming is one of the essential elements that is necessary for a person to achieve fluency with information technology. NSF researchers suggest that programming instruction should not be limited to students in computer departments, noting that learning to program has been shown to have benefits ranging from teaching general purpose problem solving and thinking skills, to helping students appreciate and understand how computers work [3].

There is an increasing amount of literature on new innovative techniques on teaching computer programming [4, 5]. Since the 1960s, with the birth of the first computer science departments, research has been active on the topic of the challenge of training students to write programs in this rapidly growing discipline. In addition, various organizations have issued reports covering such issues as computer science in liberal arts colleges [6, 7], the structure of the introductory course [8,9], standards

for accreditation [10], and secondary school curricula [11]. A particular feature of programming reinforced by Dijkstra that it is "problem-solving intensive" [12].

Programming involves multiple processes. In its simplest level the specification needs to be translated into an algorithm, which then is translated into programming code. In this paper we focus how an introductory computer course involving algorithmic problem solving techniques emphasizes critical thinking and logical reasoning to liberal arts students. This report begins by considering what Computer science is and what goals are suitable for the study of an introductory Computer science course in a liberal arts setting. We then describe the course materials and approach which integrates these ideas and can contribute to the foundation of an excellent liberal arts education. Section 2 describes the discipline of computer science in brief. Section 3 discusses the introductory computer science course taught at Eastern Connecticut State University (ECSU). Section 4 identifies the importance of the algorithmic problem solving scheme to foster logical and critical thinking in students' mind in the process of designing solution to problems. Section 5 depicts an example of a lab experiment which students conduct. Section 6 demonstrates how students analyze the performance of algorithms. Section 7 discusses student responses and section 8 draws the conclusion.

2. DISCIPLINE OF COMPUTER SCIENCE AND PROGRAMMING

Computer science is the study of algorithms and data structures [6]. One of the tasks of computer scientists is to design and develop algorithms to solve a variety of real world problems. Amongst others, the process of design may include:

-Designing correct and efficient algorithms in its mathematical and formal properties.
-Constructing hardware components to execute the algorithms.
-Programming in languages and transforming the algorithms into the statements of those languages.
-Developing productive and efficient application software.

Among the above four operations an introductory course often focuses on just the first and the third items.

2.1 Algorithm and Pseudo-code

Finding a solution to a given problem is called algorithm discovery. It is the most challenging and creative part of the problem solving process. The dictionary defines the word as: *al·go·rithm, n. A step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps.* The word algorithm is derived from the last name of Muhammad Ibn Musa Al-Khowarizmi (a.d. 780- 850?), a famous Persian Mathematician of the eighth and ninth century [13]. His last name was rendered as Algorismus in Latin characters, and the formalized procedures that he initiated became known as algorithms.

Generally computer scientists use a notation called pseudo-code (false code), a set of English language constructs, to design and represent algorithms. One of the nice features of pseudo-code is an apprentice in programming including liberal arts undergraduate students can adopt it into their own personal way of thinking and problem solving as it is highly readable, has virtually no grammatical rules and resembles statements of the programming language which do not run in a computer. The advantage of introducing and emphasizing an algorithm before beginning with the actual coding is that students are able to concentrate on the problem solving steps, instead of concentrating on the complexity of the syntax of the programming language. For beginners, programming language syntax can be very discouraging and intimidating. At this stage of learning the goal should be capturing the essence of solution design instead of focusing on the complexity of the programming language implementation. Students' interest may be withdrawn if they are introduced to the syntax at the very beginning.

3. MATHEMATICS, COMPUTER SCIENCE AND LIBERAL ARTS

Ideas from Mathematics are an integral part of Computer science in multiple different ways:

-Mathematical reasoning, logical proofs and assertions provide means to prove the correctness of algorithms, programming language constructs and syntax, hardware precision.

-Analysis of algorithms can be carried out using mathematical tools, for example: probability, statistics, etc.

-The objects of mathematics for example sets, trees, graphs etc provide the foundation for models in Computer science.

Moreover mathematics is intertwined in many subjects supported by computing: economics, biology, psychology, physical science, geology, etc, to name a few disciplines that are taught in author's institution.

3.1 Introductory Computer Science Course at ECSU

"*CSC 110 – Introduction to Problem Solving with Computers*", seeking an introduction to computer science and how it is applied to problem solving, serves as a basis for both beginning computer science students as well as a general education course for liberal arts students at ECSU. The course begins by showing that computer science is the study of algorithms, which is the central theme of the course. Algorithmic problem solving techniques are taught by introducing methods for designing and representing algorithms. The course also covers Boolean logic and gates, circuit construction algorithm, computer organization, etc. One major area under discussion for this course is the selection of mathematical and theoretical concepts and techniques inherent to computer science. To keep the students interested and engaged it is important to show how new ideas and theory are constantly joined to computer applications with a view to solve problems. Lab assignments are used to demonstrate applications for most topics which help students develop empirical skills that are common to all scientific endeavors. By studying and investigating applications in respective fields students will learn to apply their programming skills to a wide variety of problems.

4. IMPORTANCE OF ALGORITHM PROBLEM SOLVING SCHEME

One may ask a student: why do we need to write a program? The answer is that the computer program provides a solution to a problem. An existing problem is brought to the attention of the programmer, and the programmer offers a solution by writing a program. In the process of programming a given problem is first translated to a specification and then to pseudo code. Writing pseudo code from a specification could be challenging. Students get involved to think critically while creating the solution steps using algorithmic problem-solving techniques.

In the existing research in computer education much has been mentioned focusing on new and interesting ways to teach programming. Even with this innovation there are some aspects of program design that are persistent. No matter how small a program is, each and every program should have the basic structure with three main parts as follows: Input Data, Process Data, Print Result. Moreover, the steps of Program Development Life Cycle (PDLC) do not change. PDLC provides an organized plan of breaking down the complete program development into manageable tasks. Each of the tasks needs to be completed before one proceeds to the next phase. The phases are namely: defining the program, designing the program, coding, testing and debugging, documentation, implementation and maintenance. As the focus of the paper is on the beginning programmer, we omit the maintenance part in our course.

During this phase of the course, students will be learning to analyze problems and then think logically to formalize their thoughts in the process of solving a problem. Students learn to write their thoughts into steps of algorithms and create algorithms for the simplest problems that they face in their daily lives; for example: adding two digit numbers, finding the largest value in a list, searching for a given name in a list, etc. The next section illustrates laboratory experiments set up to demonstrate the execution of an algorithm written only in pseudo-code.

5. LABORATORY EXPERIMENTS

The discipline of computer science is empirical as well as theoretical. Learning comes not just from reading about concepts like algorithms, but manipulating and observing them as well. Laboratory experiments are designed where students acquire hands-on experience through algorithm animation. The labs introduce the concept of algorithms, in which one can observe an algorithm being executed and watch as data values are dynamically transformed into output result. Bringing an algorithm in life through animation can help students understand what the algorithm does and how it works. In lab experiments they repeatedly investigate how the steps of a previously written algorithm can be executed one at a time. Figure 1 depicts a screen shot for laboratory animation of a search algorithm [14]. The steps of the algorithm are written in pseudo-code. Students work through implementation of steps of pre-written algorithms and calculate solutions for given input data. This phase benefits students on seeking solutions to problems using critical thinking without going through the complexity of learning structured-code at the very beginning of the semester. In Figure 1 with the *Reset* button a new set of data values can be generated at random. By clicking *Input Data Set* button one can input any special set of data to observe how the algorithm performs on the given set of the student's choice. If the *Run* button is clicked, the animator executes the entire algorithm, pausing only for input and output. The *Step* button is used to just execute a single step of the algorithm one after another. The students will primarily use the *Step* mode as the main goal here is to gain insight into the nature of the algorithm and gather empirical data. The slider bar adjusts the speed of execution.

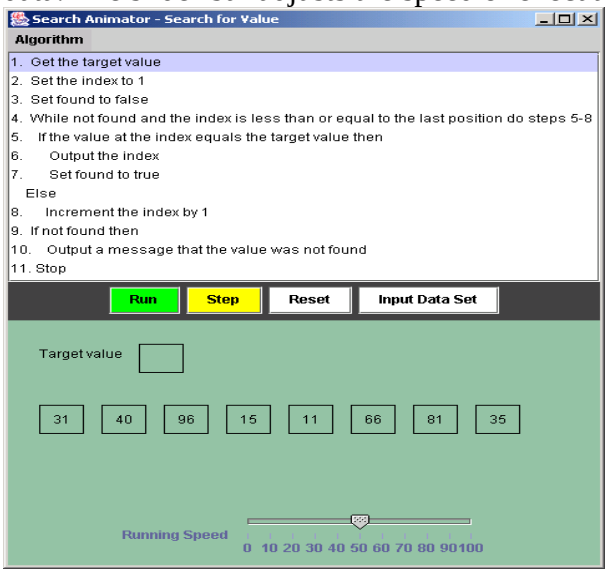

Figure 1: Screen shot of the '*search a value in a list*' algorithm.

As the execution progresses, a visual signal is given to the user by shading the next step to be executed. This enables one to keep track of where the algorithm is in execution and what is happening to the data

as the step is being executed. As an exercise the students modify their *Target* value and record exactly how many times each of the steps is executed. The objectives of these labs are not only to solidify the understanding of an algorithm by working through the animator, but also to analyze algorithms as computer scientists do. They also learn to obtain a general mathematical formula about the performance of the algorithms.

6. PERFORMANCE ANALYSIS OF ALGORITHM

Computer scientists design algorithms to solve problems and there is a set of desirable properties of algorithms which includes, ease of understanding, elegance, and efficiency in addition to the correctness. A problem can be solved in several different ways. Computer scientists strive to design the most efficient ones under given constraints. In this course students perform laboratory experiment with several search algorithms and investigate which algorithm performs better under which conditions. The performance analysis of algorithm (e.g., the study of the algorithm's efficiency) is an essential part of computer science. One way to find out the time efficiency of an algorithm is to examine its work done. In order to investigate the performance of the sequential search algorithm students derive a formula for the average number of comparison done by the algorithm. In this algorithm comparisons are made in Step 5 (see Figure 1). In general, the target value is likely to be in any one position in the list as in any other position. One way to measure the number of comparisons is to compute the average of the number of comparisons required for different positions of target. Students construct a table to record the number of comparisons made by the algorithm as the position of the target value changes. The values for the table are computed by running the algorithm several times and keeping track of the number of times step 5 is executed as the target value changes. For a list of size k the average number of comparisons needed can be calculated as follows:

$C_{avg} = (1 + 2 + 3 + ...k)/k$ or, $C_{avg} = (k+1)/2$

From this formula one can conclude that the average number of comparisons increases linearly with the size of the list. Thus students are able to analyze algorithms to obtain measures of performance in terms of the size of the problem to be solved.

There are animations for other algorithms namely, *search for the largest, binary search, selections sort, searching for a particular pattern of characters within a segment of text (pattern matching).* Once they have enough understanding about different searching and sorting algorithms, they are asked to compare the relative performances, pros and cons of each of them. We have similar lab exercises where the students are able to run animations of various practical problems. They work with the simulator to understand the operation of a simple Von Neumann machine. A lab in data encryption lets them explore the transformation of data into a secure form so that it can be accessed by only those who have authorization to do so. Another simulator models the transmission of messages across a wide area network computer networks. Students discover how errors can occur during network transmission and the steps used in detection and correction.

7. STUDENT RESPONSE TO THE COURSE

Student response to this course, in general, is enthusiastic. Course evaluation data over the past five years as shown in Table 1 depicts that 85% of the students *Agree* or *Strongly Agree* on the statement "*I would rate the overall quality of this course as high*" and 87% of the students *Agree* or *Strongly Agree* on the statement "*My experience in this class make me want to learn more about this subject*". Learning problem solving by developing algorithmic steps help organize their thoughts to arrive to a solution to any problem they encounter. Introduction to this technique increases student's interest to focus on problem solving. They are relieved to learn computer science concepts and obtain hands-on experience

executing different lab experiments without going through the rigor of writing their own program. As mentioned earlier, learning to program is not one of the main goals of this introductory course. Many students were highly satisfied to be able to write simple programs on their own with input statements, few computation statements and output statements. This outcome was very much rewarding to the instructor. Students seem to be pleased to acquire this knowledge.

Items asked to give students' impression	% of students Agree or Strongly Agree
I would rate the overall quality of this course as high	85%
My experience in this class make me want to learn more about this subject	87%

Table 1: Student evaluation data average for past 5 years.

8. CONCLUSIONS

In this paper we discussed how an introductory computer science course focusing on algorithmic problem solving techniques help liberal arts students exercise their critical thinking and logical reasoning skills. In the process of problem solving students first learn to analyze problems and then think logically to formalize their thoughts into the steps of algorithms using pseudo-code. To retain student's attention and keep them engaged, small scale laboratory experiments are designed where they acquire hands-on experience through algorithm (written in pseudo-code) animation. This phase benefits learner programmer not only on seeking solutions to problems thinking critically, but also capturing the essence of designing a solution instead of going through the complexity of learning structured-code at the very beginning.

9. REFERENCES

[1] Papert, S., Mindstorms: Children, computers and powerful ideas, New York: Basic Books, 1980.

[2] Committee on Information Technology Literacy, National Research Council, *Washington DC,* "Being fluent with information technology*", 1999.*

[3] Soloway, E., Should We Teach Students to Program, *Communications of the ACM*, 36(10), 21-24, 1993.

[4] Jenkins T., The Motivation of Students of Programming, *Proceedings of ITiCSE 2001*, pp 53-56, 2001.

[5] A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science, *ACM Journal on Educational Resources in Computing*, Vol. 7, No. 2, Article 2, June 2007.

[6] Gibbs, N. E. and Tucker, A. B., A Model Curriculum for a Liberal Arts Degree in Computer Science, *Communication of the ACM*, 29, 3, 202–210, 1986.

[7] Walker, H. M. and Schneider, G. M., A Revised Model Curriculum for a Liberal Arts Degree in Computer Science, *Communications of the ACM* 39, 12, 85–95, 1996.

[8] Koffman E. Stemple D. Wardle C., Recommended curriculum for CS2, 1984: a report of the ACM curriculum task force for CS2, *Communications of the ACM*, v.28 n.8, p.815-818, Aug. 1985.

[9] Koffman E. Muller P. Wardle C. ,Recommendations for the first course on computer science, Proceedings of the 15th SIGCSE technical symposium on Computer science education, 1984.

[10] Mulder M. C. J. F., Computer Science Program Requirements and Accreditation, *An Interim Report of the ACM/IEEE Computer Society Joint Task Force. Commun. ACM 27*(4): 330-335, 1984.
[11] CORPORATE Pre-College Task Force Committee of the Educ. Board of the ACM, ACM model high school computer science curriculum, *Communications of the ACM*, v.36 n.5, p.87-90, May 1993.
[12] Dijkstra Edsger W., On the Cruelty of Really Teaching Computing Science, *Communications of the ACM*, Vol.32, pp 1398-1404, 1989.
[13] Schneider G.M. and Gersting J. L., Invitation to Computer Science, C++ Version, Fourth Edition, Course Technology Incorporated, 2007.
[14] Lambert K., and Whaley T., Invitation to Computer Science Laboratory Manual, Thompson course Technology, 2007.