Computer Science vs. Computer Literacy
Which to Teach?

Viera K. Proulx
College of Computer Science, Northeastern University
Boston, MA, 02115, USA
vkp@ccs.neu.edu

With the widespread use of computers in all areas of work and play, it became clear that all students (in secondary schools, universities, and even in elementary schools) should be taught 'something about computers'. The dilemma is, whether to teach 'computer science', or just 'computer literacy'. In this paper we argue that both computer science and a literate use of computers need to be taught to all students, if we want them to function effectively in the new information age.

Using examples from real teaching situations, we illustrate how to incorporate the teaching of effective computer literacy skills into traditional, programming based, computer science course. In the second part of the paper we show how teaching fundamentals of computer science within the context of a computer literacy course can improve student's ability to learn how to become an effective user of computer technology.

Computers in computer science education - current situation.

In the USA, Advanced Placements courses taught in secondary schools are supposed to cover material typically taught in the first year university courses. Students who successfully pass the national advanced placement exam in a given subject, receive university credit for that subject. The curriculum for Advanced Placement in Computer Science, covers basic topics in computer science - algorithms, data structures, - in the context of programming in Pascal. There is no requirement that students be able to use spreadsheet application, build and query a data base, or even use a word processor. Similarly, the Curriculum '91 - the major document used to define university level computer science programs silently assumes that students will learn how to use computers effectively, but does not have any specific recommendations or requirements. Outside of the USA, especially in the countries, where computers are not yet a commonplace item, the situation is similar. Computer science educators shun teaching of 'computer literacy' as simple skill building and concentrate on the seminal, theoretical foundations of the field. Many students become skilled programmers in one of the higher level languages (C, Scheme, Lisp, Pascal), can build a compiler or an operating system, analyze efficiency of an algorithm using powerful mathematical methods, yet never used a spreadsheet to do a simple calculation.

Are we exaggerating? No. In the Spring 1994 I taught a graduate course on Computer Architecture to about 30 graduate students at Northeastern University in Boston. For the first assignment, in which students had to compare the performance improvement achieved through different changes in the underlying architecture I insisted that students perform the calculation and demonstrate their results using a spreadsheet application of their choice. More than half of the class has never used a spreadsheet and

did not know where to find one. These were not weak students. Our admissions standards are quite high and for most of these students this was at least their third quarter in our program. Some of them were working in jobs that required daily use of computers. Two of these students were working for our system group. Yet, they never had the opportunity to explore how computers are used today by a large segment of professionals in other fields.

Is this an isolated incident? We do not believe so. Several years ago, most of our undergraduate computer science students completed their first year without ever using computer for anything other than programming, word processing, and corresponding via electronic mail. While they may have used computer applications to study foreign language, to conduct simulated physics experiments, or learn about some topics in history, computers were not used to illustrate lessons in computer science - not beyond displaying the source code and possibly some other information supplied by the debugger.

Computer science educators rarely use computers as educational tools, or as tools that assist in problem solving, or for modeling and simulating events that are being studied. One needs to only looks through catalogs of educational software, or visit the vendor exhibits of major conferences on uses of computers in education.

Computers as educational tools in teaching computer science - new approaches.

There are two basic ways in which computers can be used to enhance teaching of computer science. The first is by using standard applications such as spreadsheets, data base programs, and even word processing programs to illustrate the fundamental concepts. Spreadsheets should be used to analyze and display graphically timing data or other data measuring the performance of different algorithms. Data base programs can illustrate the need and the use of indirect addressing, index arrays, etc. Building a simple paint program introduces student to the idea of main event loop and points out some basic problems with the user interface design. Comparing simple student solutions with commercial version of the program gives the student some understanding of the complex issues facing practicing computer scientists today.

The second use of computers in computer science should be for modeling of dynamic processes that are the basis of all computer science. well designed models of Turing machine, logic circuits, an animation of the behavior of a data base program when processing a query or making an update, interactive animations of all basic algorithms students encounter, a model of information flow between different components of computer system, - these all would help in de-mystifying the study of computer science.

Let us now present some examples how using computers as teaching and production tools can enhance the effectiveness of teaching of computer science. We start by describing a series of exercises in which students concurrently learn how to use a spreadsheet program and learn how to conduct experimental evaluation and comparison of algorithms. These exercises are used in our first year of study, after students mastered Pascal programming, including records, pointers, and recursion.

<u>Binary search exercise.</u>

The first exercises involves binary search of a sorted table. The procedure that implements the search is extremely simple, so there is little programming to be done. Procedure returns the location of the item it found (zero, if not found), and the number of probes needed to find the item. Students are asked to write the procedure, and write a user interface driver that will allow them to test the procedure thoroughly. The procedure header is predefined, and once the procedure works, students insert it into a main program that automatically generates a large number of experiments with increasing array size, and collects the data into a file. For each array size, there are several items being searched for, and the program records the minimum , maximum, and the average number of probes needed to find an item. Students are asked to use a spreadsheet application to display the table and plot the results.

Heapsort exercise.
When students program a heapsort, they are asked to count both the number of moves and the number of comparisons that algorithm needs first to build a heap from unsorted array, then to create a sorted array from the heap. Students use the binary search program as a model that teaches them how to collect data, format it so the file could be imported into a spreadsheet, and how to test the procedure independently of the data-collecting driver. They are now asked to display not only the data they collected, but also the curves representing the theoretical bounds on the algorithm complexity that have been derived in the class. An example of the results is shown below.

Sorting algorithms comparison.
Students do no programming in this exercise. They are given a program that allows them to collect timing data of 13 different sorting algorithms, and design the appropriate experiments (number of tries, array sizes, array data: random, sorted, inverted) and select which algorithms should be tested. Only three dimensional charts show the results in a meaningful way. The lesson here is in designing experiments that verify or contradict the hypothesis, and on experimenting with the best way of displaying the data that has been collected.

These exercises do not teach students all tricks one can do with spreadsheets. Some additional techniques are introduced in class, but mainly students now understand the power of this tool, have several examples of its use, and learn to use the help system and tutorial to learn more. They also learn that not all programming is done at the higher level language level, and that one should use the right tool for a given task.

Another example of use of computer as a teaching tool is in presenting complex algorithms. We teach students a number of different sorting algorithms because we want to illustrate the number of choices faced by a designer of any computer program. We also want to demonstrate the basic problem solving strategies, and the cost associated with their use: divide and conquer, case enumeration, recursion, backtracking; cost in terms of time, space, algorithm difficulty, etc. By observing the dynamic behavior of a given algorithm using a well-designed interactive animation, many of these points can be made more effectively. After playing with the animation for just a few minutes, students understand how insertion or selection sort works. They can see in action the behavior of divide and conquer algorithm by watching a quicksort. They can predict the next step of algorithm searching for a shortest path in a graph -

and see immediately, whether they understand the algorithm. Seeing the behavior of the algorithm, and having the opportunity to experiment with different alternatives, the opportunity to ask questions, make s learning new algorithm no more difficult than learning a new card game or board game.

Students that use computers in history courses may create a multimedia project that is a tells us about some events students were learning about. The result is visually appealing, attractive, and engaging. Students in computer science classes write programs that generate several lines of output or save the results to a file. Most of the time they do not see anything tangible, and have a hard time in finding whether the results are correct, or what may have gone wrong. By making it possible to generate sensible graphical output, students see where the program does not behave as expected. It is not only a debugging tool, but also a powerful tool for motivating students. They also learn what is behind all those exciting images computers bring to us every day. Instead of writing the first loop to add ten numbers, students draw a balloon that floats into the air (by painting and erasing a circle in different locations). They program a robot to find an item in a square grid and see if the algorithm they invented works as expected. They write a simple MiniPaint program with only four choices: circle, square, line, and stop - and learn about the main event loop, they have a nice program to show to others, and begin thinking about the difficulties encountered in user interface design.

Other opportunities lie in seeing a model of computer at all different levels, including the theoretical world of finite state automata, in exploring a multimedia tour through the computer history with tangible examples of what computers were capable of at a given point in history, exploration of the different ways used to store sound, pictures, and video images, observing the adaptive behavior of a neural net system, etc.

Is not using computers as educational tools in teaching computer science a serious problem? Does it matter how we teach computer science? We think it does! For a long time the methods used in teaching mathematics produced a significant percentage of people 'afraid of math'. Teachers explained abstract formulas, with very little connection to useful activities from daily life. Similarly today, the teaching of computer science today is mostly abstract, inaccessible to those not proficient in abstract thinking, 'too complex' to be understood by average people. It discourages many from studying computer science, and undermines the confidence of many competent users who may have been interested in learning more about computing.

Computer literacy - current situation

Today, everybody agrees that all students should learn something about computers. Most students with access to computers use them at least for writing papers and written reports. A typical course in computer literacy includes the use of spreadsheets for tabulating data and computing standard statistical functions, as well as displaying the results graphically. It also includes the use of database applications - learning how to extract information from a database, and possibly, how to define and build one. In addition, such course may cover one of presentation graphics programs, a statistical simulation package, desktop publishing package, or a CAD program relevant for student's field of study. It is becoming a norm, that every professional program of study

includes a course on use of computers relevant to the particular subject. We need to support this trend by providing instruction that will give students knowledge of computing that will transcend current trends or specific applications packages and computer systems. The knowledge student gains during such study should form a foundation of understanding of computing that will serve the students throughout their careers.

How can we increase the effectiveness of computer literacy courses? We can do it by using different applications packages as examples of computer programs, as windows into the computer architecture, or as case studies in hierarchical structure of programming languages. Spreadsheet programs allow us to experiment with loss of accuracy in iterations - illustrating the effect of limited space for number representation. Explanation of different embedded functions leads into discussion of algorithms. To use even the simplest package, students need to learn about the operating system user interface. It is important, to explain, however simply, what is happening when different selections are made, and buttons are pushed. By understanding the underlying operations, students learn how all computers are the same, and will learn what questions to ask when they are introduced to a new system.

Two examples illustrate the importance of learning what is happening inside the computer. The first one is about a secretary who used mail merge program to generate letters to a mailing list. All was great, except that not all addresses contained the same number of lines. After all the wonderful work the computer did, she was editing out the blank lines by hand. She never learned, that the computer could do it for her. Well, she would have to have done a little macro programming - but nobody ever told her. The second example is about a computer literacy class. Students were supposed to learn how to use spreadsheet and database applications on two different computer systems. At first, they were frustrated in trying to remember the right key sequences for different operations - nothing they learned about the first system seemed to make sense in the new environment. After spending some time learning about programming, and about the underlying computer architecture, they began to see the similarities between the two different systems, they understood how the computer interpreted their commands, and could easily adopt to the new system. They made fewer mistakes, needed to ask fewer questions, and were more confident not only in using the commands they studied in the class, but also in exploring other features of the system.

Computer literacy with computer science: why and how

What are the computer science topics that shout be included in a computer literacy course? The Association for Computing Machinery published recently a report entitled ACM Model High School Computer Science Curriculum. The report recommends a computer science course for all high school students, and lists the topics that should be included in such course. They are: algorithms, programming languages, operating systems and user support, computer architecture, and social, ethical, and professional context. In addition, instructors should include additional topics based on their interests, and cover examples of several areas of computer applications in today's world. Model curriculum implementations include a model based on application packages. In addition, the report describes the level of competency students should attain in different areas.

We elaborate on this list of proficiencies, in order to illustrate the need for including a particular topic in the general course of study. Students should learn that **algorithm** is a precise description of a process that will be performed by computer, human, or some machine. They learn about the basic building blocks of algorithms, how algorithms can be represented in different forms - both formal and informal, and they observe that 'some things are easier said than done' - that is, that some algorithms may be easy to describe, but impossible to carry out in reasonable time. This gives students a real understanding of the statement 'the computer will only do what you tell it to do'. will avoid a lot of frustration, and prepare students for using any computer system available to them.

When learning about **programming languages**, students get a better understanding why the command language of the operating system user interface is different from the set of commands used inside an application,. By looking at the lowest levels of computer languages (machine language compared to the language of calculators) students begin to understand the basic computer architecture, and the levels of complexity built upon it, to assist the user. Overall, it will be easier for them to learn a new language or user interface - they will know what to look for and the commands will become more meaningful.

Computers today come equipped with a plethora of different attachments: disk drives, printers, scanners, microphones, network connections, etc. Students need to understand the role of **operating system** as a resource manager, that keeps track of all the resources, creates directories of all files, with information about their physical location, logical hierarchical naming organization, size, type of organization and permissible access., etc. By learning about the problems operating system needs to address, students gain a better understanding of how the resources are organized. This empowers the students to utilize the resources better, to understand what needs to be done to move information from one computer to another, or from one format to another. Students also need to learn about the use of computers as world wide communication and information system. The resources available electronically surpass the resources of any single library student can access. Every literate citizen of tomorrow should be able to access these resources effectively.

By covering the previous topics, students already gained some understanding of **computer architecture** - the CPU, memory, I/O model. To increase student's understanding of computers, we should also look in more detail at the data representation in the computer. Students are familiar with several ways of representing information: traffic signs, airport pictographs, roman and arabic number system, alphabets used in different languages, the Morse code, to name a few. By learning about ASCII representation of characters and the binary number system students will understand better the differences between the file formats, the effect of rounding errors, the loss of accuracy, etc. The computer will no longer be a black box. It will become an effective tool to accomplish desired tasks.

As computers change almost every profession today, they also have a great impact on the social structure of the **society**. Their use brings in new types of problems and conflicts society has to learn to deal with. The issues of privacy, security, intellectual

property rights, trespassing, warranties, and others attain new meanings and interpretations in the context of computer based systems. All literate citizens need to be aware of these issues, understand the dangers, and see the need for new rules and laws governing the use of computer based systems. Use of computer systems is bringing with it a new class division - those without access to computer information network are becoming increasingly disenfranchised. We have to find the ways to narrow this gap. Teaching computer literacy with computer science to all high school students is one step in this direction.

Of course, studying additional topics and examples of applications of computers to solving problems in today's world will increase student's awareness of the enormous potential and impact computers have on us today. They will understand how important it is to continue learning about new uses of computers, new horizons opening up every day.

Conclusion and acknowledgments

We suggested two ways of improving the teaching of computer science to all students at both the secondary school and university level. The first point made is that computer science should be taught as a foundation of understanding today's information age, and a necessity for becoming an effective user of computer technology. The second point was that the methods used for teaching computer science must employ the best that computers can offer as a teaching tools. The ideas for this paper are results of author's participation in two exciting projects: the ACM Task Force of the Pre-College Committee on High School Computer Science Curriculum, and the Active Learning Curriculum for Computer Science Project at College of Computer Science at Northeastern University in Boston. The colleagues and friends working on these two projects helped in formulating the ideas in this paper and the author would like to acknowledge their contribution with deep gratitude. They are, Charles Bruen, Philip East, Darlene Grantham, Susan Merritt, Charles Rice, Gerry Segal, and Carol Wolf from the ACM Task Force, and Harriet Fell and Richard Rasala, collaborators in the project at Northeastern University.

Bibliography

1. ACM Task Force, "ACM Model High School Computer Science Curriculum", Report of the Task Force on High School Curriculum of the ACM Pre-College Committee", ACM Press, 1993.

2. Moshe Augenstein and Langsam Yedidyah, Automatic Generation of Graphic Displays of Data Structures Through a Preprocessor, SIGCSE Bulletin, February 1988, Vol. 20, No. 1, p. 148.

3. C. Brown, H. J. Fell, V. K. Proulx, R. Rasala, "Instructional Frameworks: Toolkits and Abstractions in Introductory Computer Science", Proceedings of ACM Computer Science Conference, Indianapolis, IN, February 1993.

4. C. Brown, H. J. Fell, V. K. Proulx, R. Rasala, "Using Visual Feedback and Model Programs in Introductory Computer Science", Journal of Computing in Higher Education, Fall 1992, Vol. 4(1), 3-26.

5. C. Brown, H. J. Fell, V. K. Proulx, R. Rasala, 'Programming by Example and Experimentation", in Computer Assisted Learning, Proceedings of the 4th International Conference on Computers and Learning, ICCAL '92, Wolfville, Nova Scotia, Canada, June 1992, I. Tomek, ed., Springer Verlag, 136-147.

6. Marc H. Brown, Perspectives on Algorithm Animation, Proc. ACM SIGCHI '88 Conf. on Human Factors in Computing Systems, April 1988, pp. 33-44.

7. Marc H. Brown and Robert Sedgewick, "Techniques for Algorithm Animation," IEEE Software, January 1985, Vol. 22, Vol. 2, No. 1, pp. 28-39.

8. Thomas L. Naps, Algorithm Visualization in Computer Science Laboratories, SIGCSE Bulletin, February 1990, Vol. 22, No. 1, pp. 105-110.

9. Viera K. Proulx, "Computer Science/Informatics: The Study of Information World", submitted to the Sixth World Conference on Computers in Education, WCCE '95.

10. Viera K. Proulx, "Computer Science in Elementary and Secondary Schools", in Informatics and Changes in Learning, Proceedings of the IFIP TC3/WG3.1/WG3.5 Open Conference on Informatics and Changes in Learning, Gmunden, Austria, 7-11 June 1993, D. C. Johnson, B. Samways, eds., North Holland, 1993, pp. 95-101.

11. Viera K. Proulx, Harriet J. Fell, Richard Rasala, Cynthia Brown, "Interactive Animations in Computer Science", in Proceedings, Frontiers in Education '93, 23rd Annual Conference (Engineering Education: Renewing America's Technology), November 6-9, 1993, Washington, DC, IEEE Press, pp. 786-790.

12. Richard Rasala, Viera K. Proulx, Harriet J. Fell, "From Animation to Analysis in Introductory Computer Science", n Proceedings of ACM Computer Science Conference, Phoenix, AZ, March 1994, pp. 61-65.

13. James Robergé, Creating Programming Projects with Visual Impact, SIGCSE Bulletin, March 1992, Vol. 24, No. 1, pp. 230-234.

14. John T. Stasko, Tango: A Framework and System for Algorithm Animation, IEEE Computer, September 1990, Vol. 23, No. 9, pp. 27-39.

15. Allen Tucker, Fundamentals of Computing I: Logic, Problem Solving, Programs, & Computers, McGraw Hill, 1992.

16. Allen B. Tucker, et. al. (ed.), Computing Curricula 1991, Report of the ACM/IEEE-CS Joint Curriculum Task Force, ACM Press, 1991.