Lifting the Hood of the Computer:*

Program Animation with the Teaching Machine

Michael P. Bruce-Lockhart *and* Theodore S. Norvell

Electrical and Computer Engineering
Faculty of Engineering and Applied Science
Memorial University of Newfoundland

Abstract:

The teaching of computer programming concepts is hampered by the difficulty students have in visualizing the dynamic processes that are controlled by the static texts of computer programs. This is no surprise, as the students have never actually seen these processes.

To reveal what is happening "under the hood" of the computer, we have developed a new tool for program animation: the Teaching Machine. It shows an abstraction that captures some of the ways high-level programmers think of machines, by modeling aspects of both the underlying processor and the compiler. As a program executes, the Teaching Machine can show the flow of control through the source code, the evaluation of expressions, and the changing values of data objects in the memory.

The Teaching Machine allows considerable flexibility. Views that are not relevant to an example can be hidden. Execution steps can be as large as a complete subroutine call or as small as a single arithmetic operation. Memory can be viewed in any of four different formats, including a box and arrow representation, which allows automatic animation of algorithms on data structures such as linked lists and trees.

We have used the Teaching Machine in a number of ways: as an animated blackboard for an instructor to use in the classroom; as an application that students can use to investigate either canned examples or their own programs; as an component in a web tutorial; and as the centrepiece of a series of tutorial videos. The Teaching Machine has been used in a first course on programming, a second course on programming, and a course on data structures.

The current version of the Teaching Machine supports a usable subset of C++. A version that also supports Java is under development. The Teaching Machine itself is written in Java for ease of distribution over the World Wide Web.

Introduction

The instructor in the classroom faces an interesting problem in explaining programming and programming language concepts to beginning students. It is easy to show the students programs or pieces of programs, such as expressions and statements, but it is harder to show the effect of these static texts on the elements of the computer or compiler.

Students need to construct for themselves a conception of algorithmic processes and a connection between the program text and these processes. The execution of computer programs needs to be demystified, so that program design can become less ritualistic and more based on reason and understanding.

To aid with this problem, we have created a new pedagogical tool, called the Teaching Machine. The Teaching Machine provides the student with a glimpse at what is happening inside the machine, as a high-level program is being executed. It can be thought of as an interpreter, with visual displays of the state of the memory and other resources.

The Teaching Machine shows an abstraction of the computer and the compiler. We do not attempt to model the way that high-level languages are implemented in detail, for example we do not worry about machine code, even though we are illustrating languages that are normally compiled. Nor do we make the usual distinction between run-time and compile-time. These issues are important for students to understand once the basics of the language are understood; until then they are not necessary and just get in the way.

A Tour of the Teaching Machine

Figure 0 shows the Teaching Machine running as a standalone application. Within the main window are a number of subwindows, each of which presents some aspect of a C++ program as it is being executed.

* Published in *Proceedings of the Canadian Electrical and Computer Engineering Conference 2000,* Robert W. Creighton Ed., pp. 831–835, Halifax, Canada, May 2000.

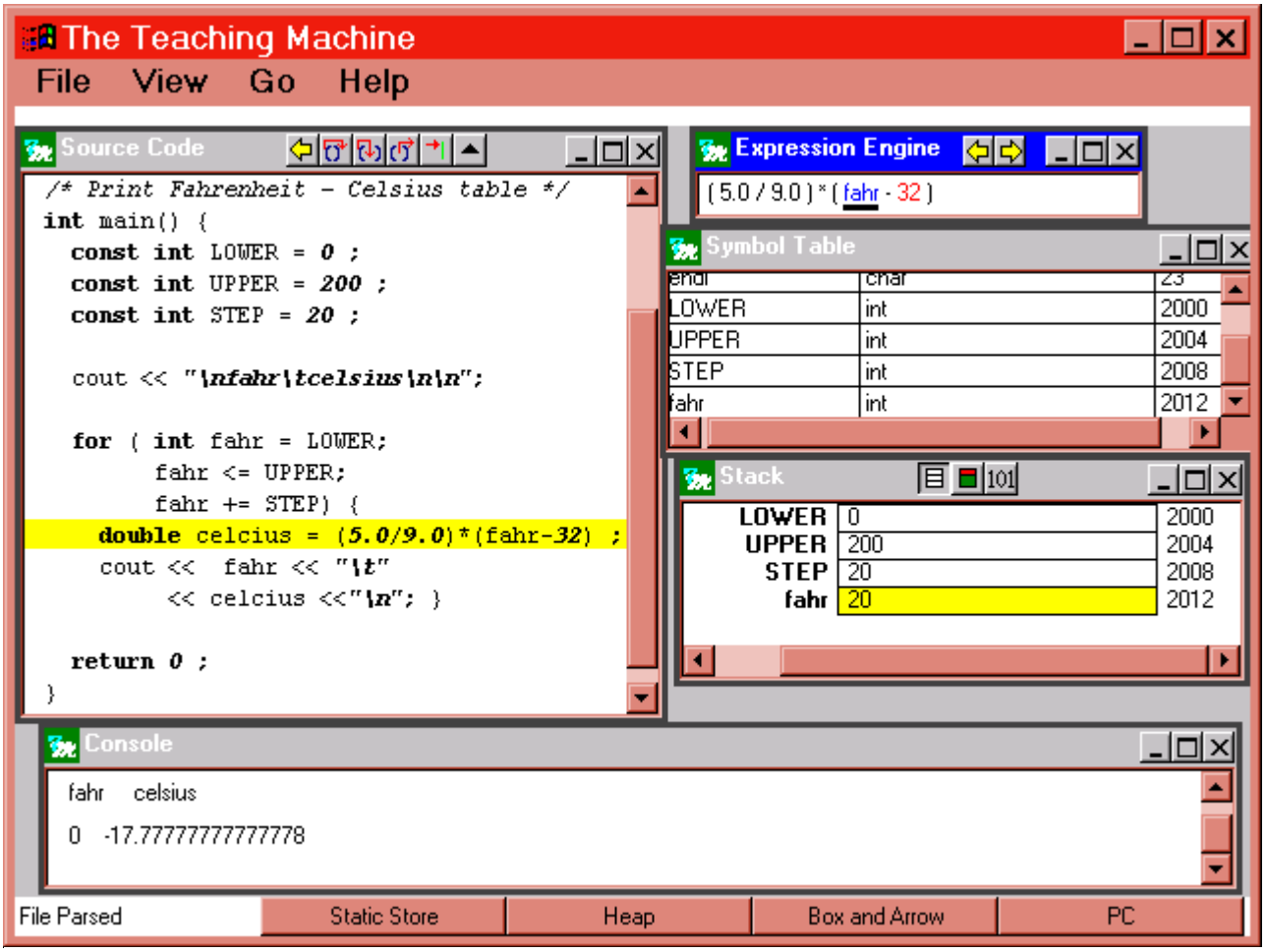


Figure 0

The subwindow labeled “Source Code” shows the text of the program. The line of code currently being executed is highlighted with a yellow background. The various buttons at the top of this subwindow control execution of the program. The program can be stepped either one expression at a time or until the program reaches a breakpoint specified using a cursor. When executing one expression at a time, subroutine calls may be stepped into or stepped over.

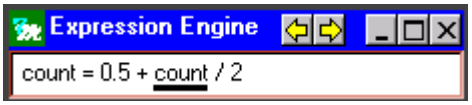
The subwindow labeled “Stack” shows the values of variables in stack region memory. There are also subwindows, not shown in Figure 0, to show the static and the heap regions of memory. The variable named “fahr” is shown highlighted because it is about to be accessed. By default, memory values are shown in a high-level representation; it is, however, possible to display them in binary. Compound variables, such as arrays and structures, can be expanded to show their members.

The subwindow labeled “Symbol Table” shows the names, types, and addresses of each variable, currently alive.

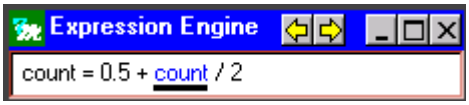
The “Console” subwindow shows the standard input and output streams, input and output are colour coded.

At the heart of the Teaching Machine is a subwindow labeled the Expression Engine. Seamlessly combining the notion of an Arithmetic Logic Unit with the compiler's expression parsing, it allows students to dissect the evaluation of expressions. The button marked with a ➪ allows the expressions to be single stepped. Figure 1 shows successive states of the expression engine as an expression is evaluated. In this example, “count” is an int variable and so the example illustrates some of the tricky aspects of C++’s arithmetic rules: that the quotient of two integers may not be what the students expect, that integers are converted to floating point numbers when combined with floating point numbers, and that floating

(a)



(b)



(c)

(d)

(e)

(f)

(g)

(h)

(i)

Figure 1

point numbers are truncated when assigned to integral variables.

At each point, the next part of the expression to be evaluated is underlined. At points (a) and (f) the next action is to look up the variable in the symbol table, so the corresponding symbol table entry is highlighted. At points (b) and (h) the next action is to access or change memory, so the corresponding memory locations are highlighted. The transitions from (d) to (e) and from (g) to (h) show implicit type conversions.

Colour is used to emphasize the process of expression evaluation. Black is used for unevaluated expressions, red for values, and blue for lvalues whose address has been calculated. The underlined occurrences of "count" in points (b) , (g), and (h) are blue.

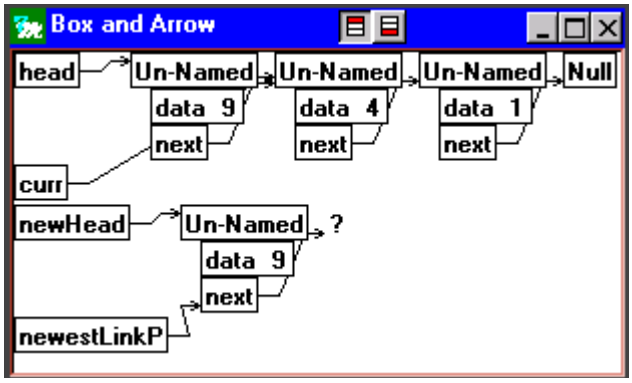


Figure 2

Because instructors or students may wish to repeat some part of an execution—perhaps to go over a large step in more detail—the Teaching Machine supports an infinite undo facility, which backs up the state of the evaluator to the state prior to the previous user interaction.

Parameters such as the size of the window; the choice, size, and arrangement of subwindows; and other display matters, such as colour schemes and fonts are all configurable by the user. This allows the instructor to choose and focus on only the aspects of execution that are most relevant to a particular example. Configurations may be saved and later reloaded.

Data Structures

In the Stack, Heap, and Static subwindows, data values are arranged according to their addresses and pointer values are displayed as integers (in decimal). This low-level view helps to demystify the idea of pointers and clarify the difference between pointer variables and pointer values. We also provide a higher-level view of memory, suitable for illustrating the manipulation of linked data structures such as linked lists and trees. Figure 2 shows a screen dump of our preliminary version of this subwindow. Stack variables are arranged in order of address on the left, while heap variables are arranged on the right.

The layout of this subwindow is entirely automatic, requiring no annotation of the source code.

In this way, the Teaching Machine, like the systems reported in [2] and [4], provides a highly automated system for algorithm animation. In contrast most existing systems for algorithm automation require some mark-up of the code (e.g., [0]), or do not use the code at all as a basis for the animation (e.g., [3]).

Modes of Use

In the Classroom

We originally conceived of the Teaching Machine as an adjunct to classroom lectures. The traditional tool of classroom instruction—the blackboard— can be used in two ways to illustrate algorithmic processes. The blackboard can be used as a stand-in for the computer's memory: In this case the instructor is frequently erasing and redrawing the various memory locations on the blackboard. Students can easily lose track of where in the algorithm the instructor is and those trying to take complete notes will be frustrated. The second way the blackboard is typically used is to write traces of the algorithm. This can lead to a lot of rewriting and can also leave students confused about the connection between the points in the trace and the points in the algorithm.

The Teaching Machine automates the first mode of instruction—implementing a sort of electronic blackboard. Since the execution of the algorithm is shown in the Source Code subwindow and the Expression Engine subwindow at the same time as the memory is shown in the Stack, Static, and Heap subwindows, students should be less prone to confusion about when actions on memory are taking place with respect to the text.

Video Presentations

It is possible to lecture on introductory programming using almost no visual aids other than the Teaching Machine. An effective extension of the classroom use of the Teaching Machine is to produce tutorial versions of demonstrations as videos. These tutorials are produced by combining direct capture of the Teaching Machine's window with a voice overlay. We have produced a number of such videos for use in a first course on computer programming. This series was placed on a CD-ROM and made available to Engineering students at Memorial taking their first programming course. A fair number of students were willing to pay ten dollars for the CD-ROM. We have yet to collect data on the perceived or actual benefit to the students.

The intended use of this series of videos is as a tutorial supplement to the regular classroom lectures. However, such videos can be used more centrally in distance education and self-study.

World Wide Web

The Teaching Machine can be run either as a stand-alone application or as an applet. As an applet, it can appear either in a window of its own, with its own menu, or within a browser window.

A number of commands for controlling the applet are available to the web page designer. For example, there are commands for loading a source file from the server, for changing the configuration, and for stepping through the program.

We have used the Teaching Machine as an applet in two kinds of web pages.

Student controlled use of applets

When the first kind of web page loads, it commands the Teaching Machine, which is already running in a different window, to load a source file from the server. The text on the web page describes the program and the main points that it illustrates. The student is then free to step through the program using the Teaching Machine in whatever manner they feel fit. This method of delivering content on the web is highly unconstrained and involves the student as an active learner who must use their ingenuity and understanding to control the Teaching Machine in such a way as to illustrate the teaching point. Such a mode of interaction is suitable for more advanced students.

Page controlled use of applets

The second kind of web page is much more constrained. In this case, the Teaching Machine appears on the browser window in one frame, while explanatory text appears in a second frame. As the user pages through the explanatory text, commands are sent to the applet to step through the program. The extent of each step can be completely controlled by the web-page designer. Such web pages can be combined with self-checking quiz questions to add engagement to the experience of the web tutorial. One drawback of this form of web page is that the student's eyes are required in both frames at once. This drawback can be overcome by the optional use of sound clips to replace the explanatory text.

Language Support

The currently released version of the Teaching Machine supports a subset of C++, corresponding roughly to the C language. Support for member functions and overloading will be included in a future version.

An effort was made in the design to isolate the language dependant aspects to a few modules so that other language interpreters can be added in the future. A version that supports Java is under development.

Experience Report

Over the past three years have used the Teaching Machine in the Faculty of Engineering at Memorial University in two courses—one, our first course in C++ and second programming course—the other, a course in data structures. The Teaching Machine was used in lectures, and the examples used were made available on the World Wide Web for student directed use with the applet.

Recently our curriculum changed to use C++ in our first programming course. The instructors are again using the Teaching Machine in the classroom. In addition, a number of tutorial videos were made available via CD-ROM or the web.

Feedback from instructors other than the developers has been positive and has lead to some improvements. Feedback from our students will be sought.

Future Directions

Although the Expression Engine separates the Teaching Machines from debuggers, the original animations still have much of the flavour of those everyday coding workhorses. The addition of the higher level, linked view for the dissection of data structures took us in the direction of algorithm animation. The introduction of classes suggests new animation techniques. Object-oriented programming is a different and much more complex beast. The challenge is to represent the interplay of objects and classes in a way comprehensible for the beginner within the bounds of a very restricted piece of real estate. We're working on ideas but they have not gotten beyond the blackboard stage as yet.

The Teaching Machine can be extended to accommodate views such as histograms and scatterplots for arrays, and circle-and-line diagrams for graphs. Such views are useful for animation algorithms, such one might find in a course on algorithms and complexity [0], [1], [3], [5]. Such views will require annotation of the source program. Although we have thus far avoided annotation, it should not be hard to add, nor should it be difficult to add new views to the current framework.

C++ is a complex language. Once its core has been captured, there will always be a demand to extend the Teaching Machine's capabilities into every corner of the language. While this is not difficult in principle, it will require significant resources. The same comment can be made with regard to other languages. A preliminary Java version is under development. Given Java's tighter structure and growing popularity, the case for a full version is even more compelling.

Once a language is fully or almost fully covered it becomes tempting to encourage students to use the Teaching Machine on their own programs. This entails greater robustness in handling illegal code and code that falls outside the supported subset than is currently implemented. So enhanced error reporting is a future goal.

Conclusion

The Teaching Machine provides the instructor a flexible educational platform to illustrate many of the concepts that perplex early students of programming. It lifts the hood of the computer in the sense that it lets the student look inside the computer to see first-hand the behaviour specified by a computer program. We feel that it can offer the instructor, either teaching in the classroom or developing multimedia instructional material, a valuable tool with which to reveal the secrets locked inside the machine.

References

[0] Marc Brown, *Algorithm Animation*, ACM Distinguished Dissertation Series, MIT Press, 1988.

[1] Arturo Conception, Lawrence Cummins, Ernest Moran, and Man M. Do, 'Algorithma 98: An Algorithm Animation Project,' *Thirteenth SIGCSE Technical Symposium on Computer Science Education*, 1999.

[2] J. Haajanen, E. Pesonius, E. Sutinen, J. Tarhio, T. Teräsvirta and P. Vanninen, 'Animation of User Algorithms on the Web,' *Proceedings of VL'97*, 1997.

[3] John Hewson, Wendy Doube, and Michael Calagaz, 'A Multimedia Animated Simulation Generator,' *The 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'99*, 1999.

[4] R. Sangwan, J. Korsh and P. LaFollette, 'A System for Program Visualization in the Classroom', *The 29th SIGCSE Technical Symposium on Computer Science Education*, 1998.

[5] Linda Stern, Harald Søndergaard and Lee Naish, 'A Strategy for Managing Content Complexity in Algorithm Animation,' *The 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'99*, 1999.