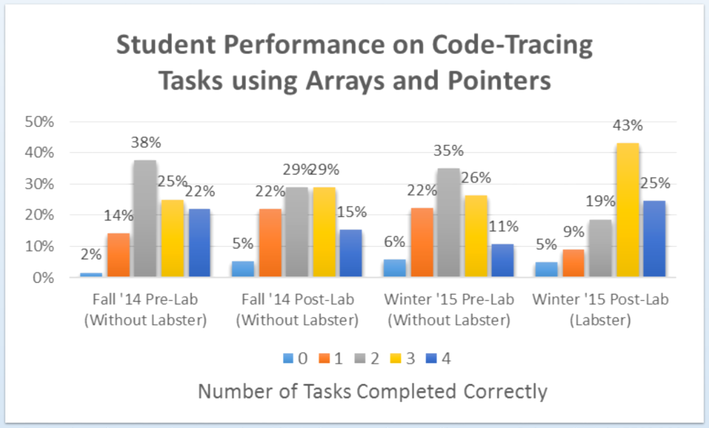
Labster: A Web-Based Tool for Interactive Program Visualization in EECS 280

James Juett (jjuett@umich.edu) and Georg Essl (gessl@umich.edu)

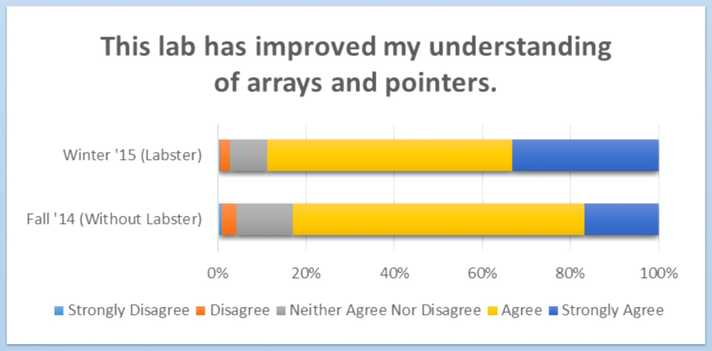
eecs280labster.eecs.umich.edu



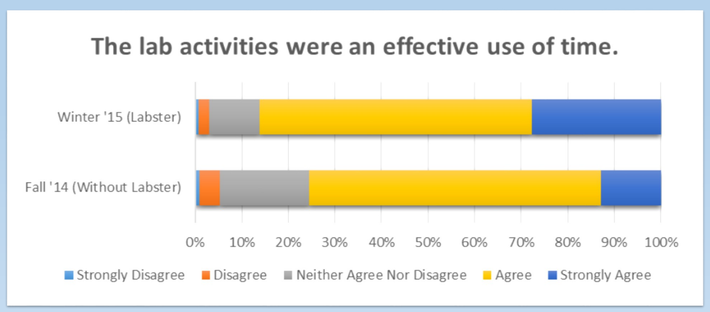
Evaluation of students' code-tracing skills before and after the "Arrays and Pointers" lab in EECS 280. Students tested after using Labster had significantly improved scores over those tested before the lab (U=6957, z=4.161, p<0.0005) and those tested after completing the same lab without Labster (U=2271, z=2.696, p<0.007). Students tested before/after the lab without Labster did not receive significantly different scores (U=1674, z=-1.123, p<0.262).

Methods and Experiment

We conducted a between-subjects experiment to evaluate the impact Labster has on students' learning. As part of the regular curriculum for the EECS 280 course at the University of Michigan, students work through interactive "lab-style" exercises during discussion sections. The "Arrays and Pointers" lab requires students to write short functions to traverse and manipulate arrays using pointers. Students in the Fall 2014 and Winter 2015 terms completed the same exercises, but the latter group used Labster to write and run their code. We compared student performance on conceptual questions and responses to survey questions both before and after the lab in each term.



Students who used Labster agreed more strongly. (U=89203, z=5.004, p<0.0005).



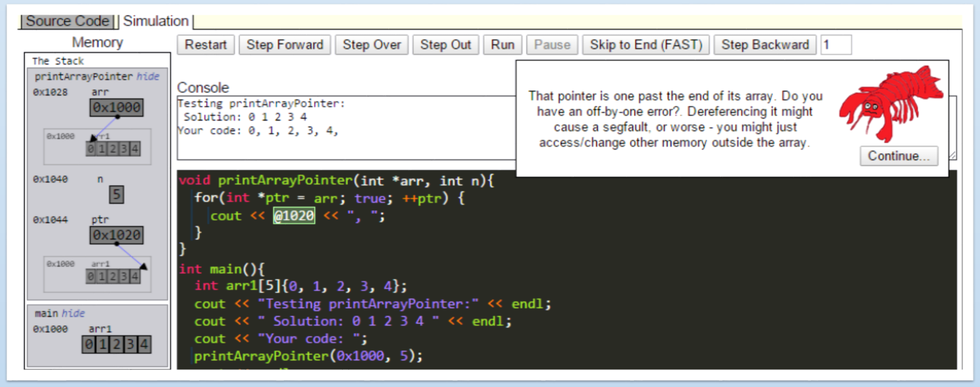
Students who used Labster agreed more strongly. (U=90674, z=5.526, p<0.0005).

What is Labster?

Labster is a web-based, interactive program visualization system designed for use in introductory programming courses. Students write their own code and Labster provides an interactive visualization that illustrates how the program would execute on a computer. Both the flow of code and the contents of memory are visualized in a natural way, and each individual step in the evaluation of expressions is cleanly animated. Students have several options for navigating through their program's execution, moving both forward and backward in time. Labster also offers educational feedback based specifically on the code each student writes and the behavior of that code at runtime.

Background and Motivation

An alarming number of students are not competent programmers after completing initial programming courses [1]. Being able to think through basic programming activities is a prerequisite for high-level program composition and problem solving, but students perform poorly on tasks that require them to mentally trace program execution [2]. One leading explanation for this is that students may not have a viable mental model of the *notional machine* that bridges the gap between written source code and the way a program actually runs [3]. Labster addresses this problem directly by illuminating the notional machine and allowing students to look inside the "black box" that runs their programs.



Labster's simulation interface. The student's program erroneously tries to dereference an out-of-bounds pointer, and Labster explains the problem. When run in a regular setting, the student's program would either crash or behave incorrectly, but with no clear indication why.

Why Does it Work?

Labster does away with the "black-box".

Students can see *how* their program works and exactly what each piece of code actually *does*.

Labster actively engages students.

Running a program is no longer a passive process. Engagement is key to effective learning. [4]

Labster is easy to use.

Using Labster is as simple as visiting a web page and there are no barriers for students.

Labster is designed for education.

Unlike regular compilers, Labster's foremost design principle is helping students learn.

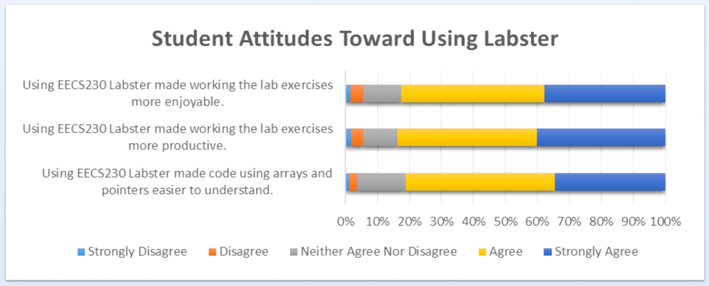
References

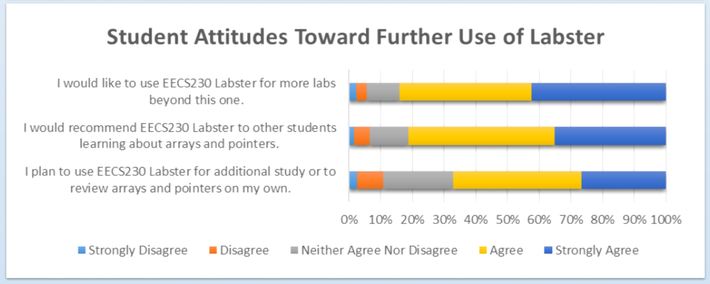
[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," ACM SIGCSE Bulletin, vol. 33, no. 4, pp. 125–180, 2001.

[2] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä et al., "A multi-national study of reading and tracing skills in novice programmers," ACM SIGCSE Bulletin, vol. 36, no. 4, pp. 119–150, 2004.

[3] J. Sorva, "Notional machines and introductory programming education," ACM Transactions on Computing Education (TOCE), vol. 13, no. 2, p. 8, 2013.

[4]C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," Journal of Visual Languages & Computing, vol. 13, no. 3, pp. 259–290, 2002.





What are Students Saying about Labster?

"Labster is great. Truly the best tool we have at our disposal for learning topics such as what has been taught so far in EECS 280. This would have been a nice tool to have in EECS183."

"...I'm a visual learner, so the representations of memory are very very helpful!"

"...I wish I was shown this thing when I started coding..."

"It was incredibly helpful for the Labster software to tell us when a function was tail-recursive or not and give explanations as to why it was tail-recursive or not—this helped elucidate the difference between recursion and tail-recursion."

"...Going in I was a bit unsure of how to write the code, but the visual process going through the code step by step and seeing what was returning and where it was going helped exponentially. Great Tool!"

"Never has a discussion sections [sic] played such a huge role in helping me master the material learned in lecture."

"...I honestly didn't understand recursion until I ran through the programs on Labster."

"...we should use it all the time for everything and I don't understand why we don't."