CodeAssessor: An Interactive, Web-Based Tool for Introductory Programming

June 22, 2012

Abstract

This paper describes CodeAssessor, a web-based assessment tool for CS1 courses that is meant to both assess student's code and to provide immediate feedback to students. The tool provides a web-based form that allows an instructor to enter a program and designate parts of the program as code blocks that are to be completed by the student. For each code block the instructor provides a set of instructions, solution code, and a scoring rubric. The instructor also provides a set of test cases for the entire program. Students complete the problem by writing a block at a time and testing their code using their own test cases. Any empty blocks are compiled using the instructor-provided code. Students may submit code blocks at any time for automatic grading by CodeAssessor. If the students code does not compile, or the students output does not match the instructors output, the student is given appropriate feedback and allowed to try again. The student is shown the instructor's code either upon correct completion of the code block, when the points available for the code block become 0, or the student gives up. This tool can be used for homework assignments, for exam problems, or for in-class assignments. Although the current language is C++, a one-line change to a couple PhP scripts is all that is required to change the compiler and execution environment to another language, such as Java. By providing the student with portions of the program, and by displaying instructor code, we hope to help students move through a problem more expeditiously and with greater understanding.

1 Introduction

There has been a great deal of recent interest in "flipping" the classroom, whereby students study lecture material before class, and then engage in more hands-on oriented activities in the classroom. One prime example is the SCALE-UP classrooms in physics, whereby lab activities are integrated with classroom instruction [1, 2]. The comparable lab activity for computer science is writing programs. However, writing a complete program as part of a lecture is usually too time-consuming of a task, even for simple programs in an introductory CS1-like course. Consequently, we have developed a web-based tool, aimed at CS1 students and named Code Assessor, that allows us to provide parts of a program for the student and allows the student to fill in code blocks to complete the rest of the program. The student can compile and test the program, and can ultimately view the instructor's model solution. This tool makes it possible to either have students view a pre-recorded lecture or to listen to a short lecture over a certain topic, and then have students complete one or more problems using CodeAssessor. Since much of the required program can already be provided, we can have students focus on just the parts that are pertinent to the current lecture topic.

CodeAssessor also has a scoring component that allows it to be used as an assessment tool for either homework exercises or for testing purposes. CodeAssessor provides a more interactive homework environment than paper-and-pencil assignments, and also provides immediate feedback for students. Students are both guided through incremental development and testing, an important technique for introductory students to learn, and if they fail to correctly write a code block in a reasonable number of attempts, they are shown the solution before moving on to the next code block. Hence students receive immediate feedback as they progress through problems, and the feedback also

ensures that they do not get stuck for prolonged periods of time. Such immediate feedback has been shown to be useful in other interactive homework assignment tools [5, 6, 3].

CodeAssessor is designed to run in any of the four major web browsers–Internet Explorer, Firefox, Safari, and Chrome. It is designed as a form-based editor because every student entering our CS1 course should have some experience with filling out forms in a web browser, and thus the barrier to entry is very low. The interface is very minimal, since the students are novice programmers and we do not want to overwhelm them with a large number of features. As designed, the interface can be explained and demonstrated to students in about 10 minutes. The tool currently supports the compilation and execution of C++ programs, but a one-line change to a couple of PHP scripts that invoke the compiler would allow the tool to be easily adapted for other languages, such as Java.

The rest of this paper further describes the tool and is organized as follows. Section 2 describes related work. Sections 3-4 present an example interaction with the tool and a brief overview of its implementation. Section 5 describes a preliminary evaluaton of the tool. Finally Section 6 concludes with a discussion of potential future work and conclusions.

2 Related Work

There are a number of existing automated problem-solving environments that are also targeted at introductory computer science students. Most of them are focused like ours, on giving the student drill work with targeted parts of a programming language. However, one important difference between our efforts and their efforts is that they hide the scaffolding code from the students, so that students only have to focus on writing a fragment of code. In working with one such system, MyProgrammingLab [8], during Fall Semester 2011, we have found that the hiding of the scaffolding code is problematic. The students are often unsure how much of the code fragment they are supposed to write, and often write too much. Their solution is then rejected as incorrect, and the students feel frustrated, because they complain that the problem description is vague. We feel that if students could see the scaffolding code, they would better recognize what type of code fragment they are being asked to write. We also believe that seeing the entire program will help students understand the context in which their code is being used.

The three online tutors that are most directly related to this project are:

1. MyProgrammingLab: This homework environment is provided by Pearson publishing [8]. Students are presented with a variety of short answer and multiple choice problems from various Pearson textbooks. Students submit their answers, which are compared against a pre-supplied set of correct answers, or alternatively, depending on the problem, the students answers may be compiled and run against a test case. Students are told whether their answer is correct or incorrect. If incorrect, the students are provided with some feedback that may help direct them toward the correct answer. Once the student correctly answers the problem, a sample correct solution is provided. As noted earlier, one shortcoming associated with MyProgrammingLab is that correct answers from the students are sometimes rejected. CodeAssessor corrects this shortcoming by compiling all answers so that any program that generates the correct output is accepted. Two additional shortcomings with MyProgrammingLab that we addressed with CodeAssessor are that 1) it does not support multi-part programming problems that require students to complete several code blocks, and 2) it cannot show the student the correct solution once the student has exceeded the maximum number of attempts. Instead the student must move onto the next problem without any idea of how to correctly solve the previous problem.

2. Practice-It!: Practice-It! [10] is a web-based application that accompanies the textbook Building Java Programs, by Stuart Reges and Marty Stepp [7]. It provides problems that give students practice with different aspects of the Java programming language. Most of the problems involve either reading a fragment of Java code, and then determining its output, or else writing fragments of Java code, and running it against pre-provided test cases. Differences between our tool and Practice-It are that 1) our tool has students write C++ code whereas Practice-It is targeted at Java users, and 2) our tool shows the scaffolding code to the students.

3. Problets: Problets is an on-line tutorial system that gives students exercises that involve reading code fragments

and then answering questions about those code fragments [4, 3]. It does not require students to write their own code fragments. Our tool emphasizes code-writing, as opposed to code-reading.

There are also more all-encompassing program assessment systems, such as Marmoset [9], that are designed to handle large programs, and that would require greater sophistication than is possessed by beginning CS students.

3 An Example Interaction

As discussed in the introduction, CodeAssessor is a form-based, web editor. The editor has a student view that allows students to complete the problems and an instructor view that allows the instructor to create and publish problems. This section describes the design of the two views by presenting an example interaction with the student and instructor views.

3.1 Student View

Figure 1 shows a sample screen shot of a student interacting with CodeAssessor. The web page begins with a brief description of the problem, and then alternates between code blocks provided by the instructor and code blocks that the student is expected to complete. Student code blocks are prefaced with a set of instructions, the number of points available for that code block, and the deduction for each attempt with that code block. At the bottom of the screen is an area for the student to enter their own test cases. At the top of the screen are three buttons that allow the student to save their work, compile and test their program, and submit their program for grading. Figure 2 shows a sample screen shot of a student's program that has just been graded. The student can also incrementally develop their program by filling in one code block at a time, and then select the "Grade This Part" button. In this case the remaining code blocks are filled in with the instructor's solution (note that the "Use Teacher Code" item has been selected for the second code block in Figure 1). The "Show Answer" button associated with each code block can be pressed at any time. If the student has successfully completed the code block or exhausted the points available for that code block, the student is shown the answer. If the student has not yet successfully completed the code block and has points available, the student is warned that seeing the answer will result in them being awarded 0 points for this code block, and asked if they still wish to see the solution.

3.2 Instructor View

Figure 3 shows the instructor's view of the tool as the instructor either creates or edits a problem. The instructor fills in blocks for the problem title and description. The instructor can then add teacher, student, and instruction blocks as appropriate to define the problem. The teacher and student blocks contain code and the instruction blocks contain instructions to be shown to the students. The code in the student blocks represents the instructor's model solution to the problem and is shown to the student via the "Show Answer" button in the student view. Finally the instructor can provide a set of test cases which will be read from stdin, or can upload a file, that allows the students to read their input from a file. There is not yet a means for students to write their output to a file, although we hope to have this functionality implemented by the beginning of Fall Semester 2012. When the instructor is ready for students to view the problem, the instructor presses the "Publish Problem" button near the top of the instructor's view. The "Toggle Student View" button allows the instructor to view the problem as a student would see it once it has been published.

4 Implementation

CodeAssessor is implemented using the Joomla Content Management system, uses mySQL and PHP on the server side, and JavaScript on the client side. The server runs on a Linux system. The mySQL database has tables that store

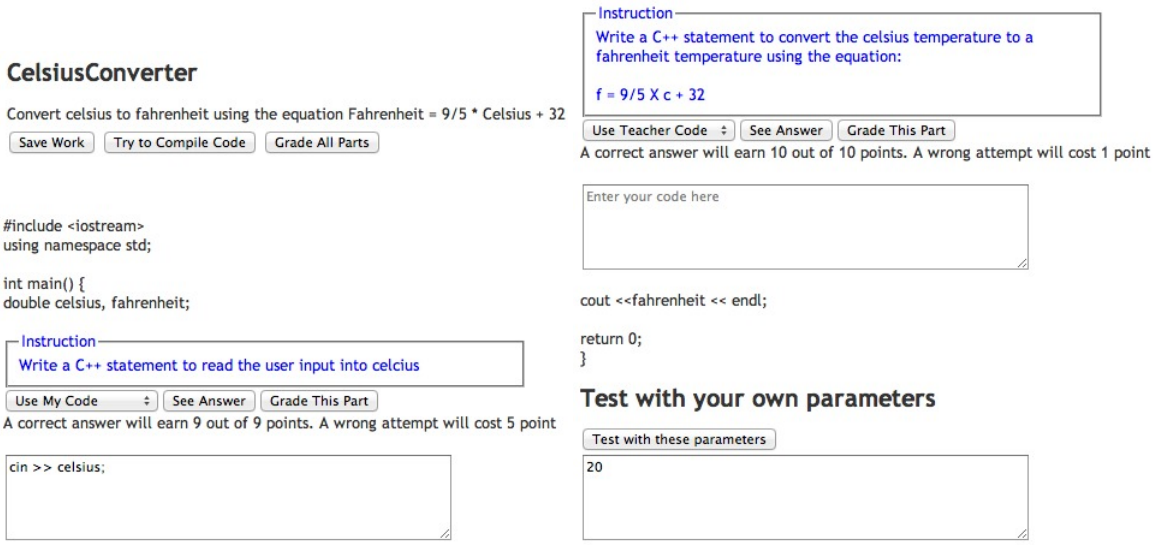

Figure 1: CodeAssessor's student view. The rightmost screenshot is the continuation of the web page shown in the leftmost screenshot.

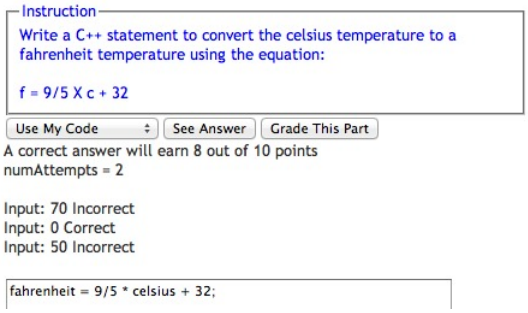

Figure 2: Diagnostic information that gets printed when a student asks that a code block be graded. In this case the student's celcius to fahrenheit conversion formula looks correct, but it uses integer rather than floating point arithmetic, and hence computes incorrect answers on two of the three test cases. The diagnostic information tells the students how many points can still be obtained by completing the code block correctly and how many attempts have already been made.

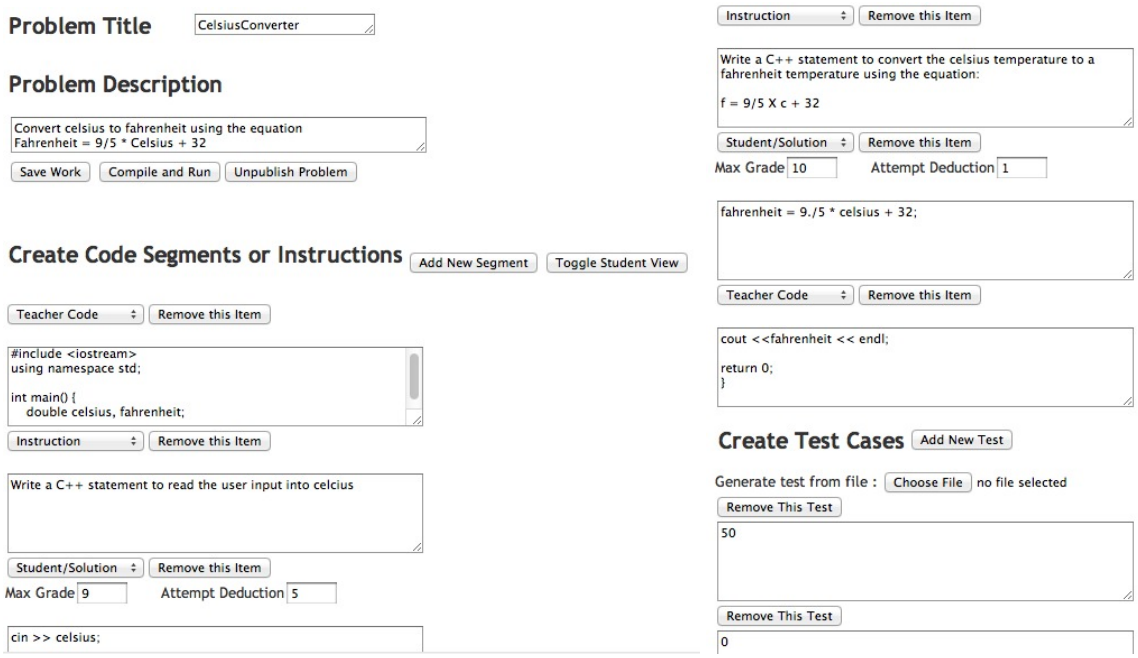

Figure 3: CodeAssessor's instructor's view. The rightmost screenshot is the continuation of the web page shown in the leftmost screenshot.

the problems, the students' solutions, and the students' grades. When the student wishes to compile a solution, the student's and instructor's code blocks are assembled into a temporary file and compiled using GNU g++. If there are any error messages, the first error message is selected and popped up in a dialog box for the student to view. Otherwise the instructor's solution is also assembled into a temporary file and compiled. The student's code and the instructor's code are then run against the test cases and the student's code is compared with the instructor's code.

CodeAssessor is sandboxed in a virtual server in order to prevent malicious attacks from compromising anything other than the virtual server. If the virtual server crashes, it is rebooted from a disk copy of the server, thus preventing any malicious attacks from corrupting the virtual server. Development for CodeAssessor is independently performed on another virtual server, and snapshots are migrated to the production server as needed.

5 Informal Evaluation

A pre-release version of CodeAssessor was used to assign an in-lab programming exercise to two different lab sections in a Spring Semester 2011 offering of CS1. Students were given a short 10 minute demonstration of the tool in which the students interactively completed two sample exercises. Students then worked with a third exercise on their own. A questionnaire was distributed to students, which they anonyomously answered after they completed the exercise.

Students indicated a preference for using CodeAssessor over paper and pencil. Students favorably cited the way CodeAssessor allows a program to be broken into parts, the easiness of compiling code, and the way that a student is allowed to look at the answer if the code block cannot be completed successfully. A couple students mentioned that they would prefer to write the full program, although that is something they will continue to do as part of their lab assignments. A couple students also felt that the interface could be improved, for example by displaying some information, such as diagnostic information, in different colors. We plan to work on incorporating color into the textual display of information in a future release of CodeAssessor.

6 Conclusions and Future Work

The CodeAssessor web tool described in this paper provides instructors with a means for assigning students partially completed C++ coding problems, automatically assessing their code through instructor-provided test cases, and providing feedback in the form of model solutions when the student gets stuck. It is a versatile tool that can be used for homework assignments, quizzes and exams, and in-class exercises. During the Fall 2012 semester we plan to use this tool in a flipped classroom, whereby students will view videotaped lectures from a prior semester before attending lecture, and will then use CodeAssessor to perform in-class exercises during lecture. We are also planning to use it for some of the class homework assignments, and possibly to use it for the coding problems on the exams. A useful future enhancement for the tool would be to find a way to generate more useful error messages for the students than the messages generated by the g++ compiler, but this might require a custom C++ implementation. This would not be as hard as it might at first appear, since the introductory CS1 course uses a very restricted subset of C++, and hence it might be possible to write a top-down parser that would generate more helpful error messages. A downside of this approach is that the tool could not be easily converted to be used with other languages, as it is now.

References

[1] BEICHNER, R., SAUL, J. M., ABBOTT, D. S., MORSE, J., DEARDORFF, D., ALLAIN, R. J., BONHAM, S. W., DANCY, M., AND RISLEY, J. *Student-Centered Activities for Large Enrollment Undergraduate Programs (SCALE-UP) project*. American Association of Physics Teachers, 2007. edited by E.F. Redish and P.J. Cooney.

[2] BEICHNER, R. J., SAUL, J. M., ALLAIN, R. J., DEARDORFF, D. L., AND ABBOT, D. S. Introduction to scale-up: Student-centered activities for large enrollment university physics. In *Annual Meeting of the American Society for Engineering Education* (Seattle, WA, 2000).

[3] KUMAR, A. N. Generation of problems, answers, grade, and feedback—case study of a fully automated tutor. *Journal on Educational Resources in Computing 5*, 3 (Sept. 2005).

[4] KUMAR, A. N. Problets homepage. http://www.problets.org/, 2012.

[5] ORSEGA, M., ZANDEN, B. T. V., AND SKINNER, C. H. Two experiments using learning rate to evaluate an experimenter developed tool for splay trees. In *SIGCSE'11 ACM technical symposium on Computer Science Education* (Dallas, TX, Mar 2011), pp. 225–234.

[6] ORSEGA, M., ZANDEN, B. T. V., AND SKINNER, C. H. Experiments with algorithm visualization tool development. In *SIGCSE'11 ACM technical symposium on Computer Science Education* (Raleigh, NC, Feb 2012), pp. 559–564.

[7] REGES, S., AND STEPP, M. *Building Java Programs: A Back to Basics Approach*, 2nd ed. Addison Wesley, 2011.

[8] SAVITCH, W. *MyProgrammingLab with Pearson eText – Access Card – for Problem Solving with C++*, 8th ed. Addison-Wesley Publishing Company, USA, 2011.

[9] SPACCO, J., HOVEMEYER, D., PUGH, W., HOLLINGSWORTH, J., PADUA-PEREZ, N., AND EMAD, F. Experiences with marmoset: Designing and using an advanced submission and testing system for programming courses. In *ITiCSE '06: Proceedings of the 11th annual conference on Innovation and technology in computer science education* (2006), ACM Press.

[10] STEPP, M., AND MILLER, J. Practice-it. http://webster.cs.washington.edu:8080/practiceit/, 2011.