WebToTeach: An Interactive Focused Programming Exercise System

David Arnow and Oleg Barshay
Department of Computer and Information Sciences
Brooklyn College
Brooklyn, NY 11210
{arnow,obarshay}@sci.brooklyn.cuny.edu

Abstract *In this paper, we describe a web-based interactive programming exercise system, aimed in part at addressing the retention crisis in CS education. The system is based on automatic program-checking software. It is designed to be easy for students and faculty to use, to support a wide variety of programming exercises, and to encourage sharing of exercises among faculty. The system is immediately available to anyone with web access with no arrangements needed. Special features such as roster maintenance can be arranged with a minimum of effort.*

Keywords: Learning technologies, retention, distance learning, automatic program checking

INTRODUCTION

Computer science education suffers from a serious crisis-- a huge fraction of students planning a CS major drop the subject within the first two semesters of study. The retention crisis is even worse for female and minority students and so limits the diversity of the CS population.

One way to address this problem is by importing teaching methods of known effectiveness from other subjects. In particular, we are interested in preparing large numbers of self-paced, highly interactive focused programming exercises that bridge the gaps in undergraduate readiness. To make this approach feasible, current web technology must be integrated into the CS pedagogy. We have done so by developing WebToTeach, a web-based, automatic homework-checking tool for computer science classes.

Students using WebToTeach can get lists of programming exercises from their instructor. Each exercise comes with a set of instructions and a form for submitting an answer. The answer may consist of one or more complete source or data files, or just a fragment of code. At the instructor's discretion, a hint or even a solution may be available. After typing an answer to the exercise in the form, the student clicks a submit button. Within a few seconds (subject to network delays), the student gets a response. If the student's answer passes all the WebToTeach checks, the answer is accepted. Otherwise, the answer is rejected. The student is provided with some information about why the answer was rejected and may possibly be given hints on how to correct it.

For students, WebToTeach provides a self-paced experience with the fundamental elements of programming, giving immediate feedback and hints at problem-solving.

The key to WebToTeach's significance is that unlike the many automatic homework checkers that have been devised for programming assignments, WebToTeach can be used to check and give instant feedback on micro-exercises, making it possible to have meaningful drill activities that help students master fundamental programming elements.

Although WebToTeach is available to anyone who comes in from the net, it is capable of maintaining individual accounts for course sections and the students and faculty associated with these. In that regard it maintains homework completion records, remembers the most recent solution attempt (so that students don't have to start working from scratch each time) and can maintain and enforce homework deadlines. Homework completion and timeliness data are provided, as is appropriate, to both students and faculty.

Currently, WebToTeach supports Java, C, C++, Fortran Ada and Pascal; other languages could be easily added to the list.

A working prototype for WebToTeach is currently used in ten or so computer science courses at Brooklyn College, providing essential material to undergraduates in elementary and intermediate courses. WebToTeach is also in the process of being integrated into instruction at other universities as well.

Besides making a difference to students in the traditional CS classroom, WebToTeach's web-based character makes it an important, perhaps indispensable, tool for distance learning as well.

RELATED WORK

A great many web-based tools have been developed. Some are conveyors of course information with creative elements such as workbook spaces and video [14] or automatic keyword indexing of questions and answers or collectors of student homework [Hsu98, Pilgrim96, Roantree98]. Others while highly creative and interactive, are applicable only to post-freshman computer science subject area, such as algorithms [Rodger96], theory [Price96] or architecture [Barker97, Connely96]. Others are more general and incorporate some degree of interactivity but only support multiple choice questions, matching, and textual fill-in that is checked by pattern-matching [Goldberg97, Medley98,]. A few of these permit numerical tolerances in checking student answers or allowed parametrized specification of exercises [Merat97, Barker97]. These limitations are part of the reason that these tools are rarely used outside the institutions in which they were developed. (This does not apply to systems such as Mallard and CAPA [Brown97, Kashy97], which are oriented to disciplines other than computer science.)

Systems that do automatic program checking have been around since the 1980s. ToTeach [Arnow95,Arnow96], a predecessor of WebToTeach at Brooklyn College, was one of a host of such tools that were developed in the late 1980s and early 1990s [Isaacson89, Kay93, REEK89, Schorsch95]. In fact, for a time it seemed that every other computer science department had one or two faculty that developed and used such a tool locally. Some of these were published but most were not.These tools varied considerably in their flexibility. Some were restricted to a particular language [Graham97, Jackson97, Mansouri98]. None were widely adopted beyond a subset of the faculty in the developer's own department. These automatic program checking systems suffered from two great weaknesses:

- They were awkward for faculty to use or very limited in the kinds of checks that could be applied to student submissions.
- They could check only programs, not fragments of programs.

The first weakness precluded widespread use, the second precluded the ability to support a new pedagogy based on large numbers of focused small exercises. WebToTeach suffers from neither of these.

One effort that is particularly noteworthy is the MIDL system [Naccache98]. Rather than defining an inflexible web-based system that other faculty must fit into, it provides a set of Java servlets, including one that does primitive program checking. Faculty can use and extend these servlets to build their own customized web-based system. Though worthy, it requires a knowledge of Java and Shell programming and so will only be useful to a handful of system builders, not to thousands of computer science instructors.

None of these tools offers the flexibility in language and exercise type that WebToTeach does.

FOUR EXERCISE EXAMPLES

The following four examples illustrate the variety of exercises that WebToTeach supports.

An exercise that involves writing a code fragment: Write an expression in C that allocates space for and returns a pointer to an array of **`n int`**s where **`n`** is an **`int`** variable containing a positive integer.

Here, the student would be given a single text area on a Web form to type

```
    (int *) malloc(n*sizeof(int))
```

or the equivalent.

An exercise that involves writing data for a test suite: Assume that a program has been written to read three integers from standard input and print the largest of them if they are all distinct and to print their sum if there are any duplicates. Submit a test suite for this program using the text areas given below. In each text area type three integers for the program's standard input followed by the integer that a correctly written program will output. Make sure you are testing for likely errors.

In this case, the student would be given multiple text areas to write his or her solution.

An exercise that involves writing a complete, single source program. Write a program that reads integers from standard input up through end of file and prints them, one to a line, in sorted order.

An exercise that involves writing several source files. Write a module (consisting of a source and header file) that provides for precise handling integers of arbitrary size. Write another module that uses this "big integer" module and that provides a factorial function and an integer exponentiation function for integer arguments of any size.

DESIGN CONSIDERATIONS

WebToTeach was designed to achieve several specific goals: educational, access, faculty collaboration, flexibility, and ease of use.

Educational

One of the remarkable differences between math education and CS education is that whereas math students are given— almost at every level— large numbers of small, focused exercises, CS homework over the course of a semester tends to be a much smaller set of exercises: "significant" (but nonetheless always "toy") programming assignments, a few thought experiments perhaps. If we are young enough, we may remember that there was one weekend in our first year in college when we had to differentiate dozens of polynomials— all pretty much the same, but afterwards we never forgot how to do that manipulation. Is there anything approaching that in CS education?

It may be that such exercises are inappropriate in our discipline. Certainly they are insufficient by themselves. However it may be the case that some of the students who lose there way early in CS education would do better if such micro-exercises were available and integrated into the curriculum.

The chore of creating micro-exercises and checking students' solutions is a serious obstacle. So a goal of WebToTeach is to support micro-exercises, to make their creation easy and to make it easy for faculty to import such exercises, developed by others, for their own students.

Web-based User Interface

The user interface had to be simple, easy to use, and familiar. This was true for students, faculty and administrators. All interactions with the system had to be possible using the web alone. Managing WebToTeach at the level of instructor or administrator should not require knowledge of any particular shell or scripting language.

Access

The system had to be accessible from the computer labs, the computer classrooms, the students workplace and home, the library and so forth. Again, that essentially means the web.

Faculty Collaboration

The system had to encourage the easy pooling of resources and the eventual accretion of a very large collection of exercises. A major goal of the project is to become a very broad CS faculty enterprise, with the contributions of dozens of faculty and the participation of hundreds more. The system therefore must scale up.

Flexibility

One of the important positive features of the original ToTeach was its flexibility. We wished to preserve that flexibility as long as it did not interfere with the above design criteria.

WEBTOTEACH SERVICES

Apart from the WebToTeach administrator, there are three classes of WebToTeach users: guests, students and faculty. Anyone can be a guest, but arrangements— though relatively minimal— must be made to create course setups for students and faculty. This setup is pretty simple actually: given a file specifying a course name, a set of faculty names and email addresses and a set of student names and email addresses, a course setup can be carried out in minutes.

Guests

Guests play the role of students but the system keeps no record of their work, will not therefore be able to let them start working where they last left off, will not enforce or inform them of deadlines, nor provide information about their activities and successes. Otherwise, they have full access to the exercises and exercise checking facilities that students have.

Students

Students are given their own distinct accounts, with usernames and passwords. They log in by selecting a course from a selection pop-up menu and typing their account information. They are shown instructor-provided message of the day information, can look at a numbered list of exercises with brief descriptions, deadline information, and submission/completion status. Clicking on such an exercise opens up a page with a complete set of instructions, links to hints and a solution (at the instructor's discretion), and an appropriate form for submitting their solutions.

Students can load into these forms their most recently submitted solution or start from an instructor-provided template (at the instructor's discretion).

Upon submitting their solution, students are told within seconds (subject to network delays) whether it has passed the instructor-provided tests. In the case of passing, their roster is updated immediately. In the case of failure, the student is given information about the cause of failure.

Faculty

The faculty interface is more complicated than that of the student, but it is still quite usable. The instructor does not need to know any scripting language, HTML or special purpose configuration file format. The interface is strictly classical-web: buttons, selections, checkboxes, and text areas. The only type of material typed into the text areas is English (or any natural language) for the student, program code, and input data for testing and expected output.

Setting up an exercise. To set up an exercise, the instructor type the name and brief description of the exercise, the exercise instructions in plain text or HTML (if desired), optionally sets a deadline, and indicates whether the student-submission will be in the form of a single "fragment" or whether it will be in multiple parts. Multiple tests can then be specified, though the way this is done depends on this choice.

In the case of fragments, the instructor specifies a test by defining a piece of code that goes before and one that goes after the fragment— the three are then concatenated to form a source file that is compiled, linked and executed with input that the instructor has provided. The output is compared to output that the instructor has specified. The instructor can direct the comparison to ignore case, and a variety of spacing issues. The pieces of code that the instructor writes can be empty— in which case the test reduces to simply testing a single source program of the student.

Other services. WebToTeach maintains roster information for instructors, allows them to forgive lateness, view the first and last correct submissions of a student, directly send mail to individual students, broadcast mail to students, set and edit the message of the day, and get statistics on homework completions.

IMPLEMENTATION

The system is written strictly in ANSI C and uses HTML only— and a relatively minimal HTML at that. Thus, CGI response is very quick and a most versions of the leading free or very cheap commercial browsers can be supported.

PERFORMANCE

The web pages in the system are designed to minimize data size, and with the CGI programs written in C and running on a reasonable machine, response time is dependent mostly on network latency. Our experience is that response is excellent as long as "ping time" is less than 100ms or so. Even with dialup connections with twice that latency, service is acceptable.

USE

WebToTeach is currently used in ten undergraduate and two graduate classes at Brooklyn College. As each semester goes by, more exercises are available for faculty to select for their own courses. As additional faculty becomes users and start creating a few exercises, the process — and the pool of available exercises increases.

AVAILABILITY

At the time of this writing, the URL for WebToTeach is http://wtt.sci.brooklyn.cuny.edu/. Anyone with one of the standard browsers can use it now as a guest. To get an account for a course, one need only send email to the WebToTeach master listed at that URL (or to the first author of this paper).

The software that supports WebToTeach is available to all who wish to run it locally on their LAN. Any flavor of Unix will do— in the not-too-distant future, NT will be supported as well.

ACKNOWLEGMENTS

We gratefully acknowledge the support of this work by the National Science Foundation (NSF CISE-9522537).

REFERENCES

[1] Arnow, D., ":-) When You Grade That: Using E-mail and the Network in Programming Courses", *Proceedings of the 10th Annual Symposium on Applied Computing*, Nashville, (March, 1995).

[2] Arnow, D.M. and Clark, D.: "Extending the Conversation: Integrating E-mail and Web Technology in CS Programming Classes", 1st Annual SIGCUE/SICSE International Symposium on Technology in CS Education, Barcelona, Spain (July, 1996).

[3] Barker, S.: CHARLIE: A Computer-Managed Homework, Assignment, Response, Learning and Instruction Environment, *FIE'97*, (November, 1997).

[4] Brown, A.L., Bransford, J.D., Ferrara, R.A., Campione, J.C.: Learning, remembering and understanding. In Kessen, W. (editor), *Handbook of Child Psychology: Cognitive Development Vol. 3*, Wiley (1983).

[5] Brown, D.J.: Writing Web-based Questions with Mallard, *FIE'97*, (November, 1997).

[6] Burris, H., and M. Darr, "The PROGRAMS Package for Integrated Grading," Program in Computing, Department of Mathematics, University of California, Los Angeles (1988).

[7] Connelly, C., Pennock, D., Bierman, A. W., and Wu, P.: Home-Study software: flexible, interactive and distributed software for independent study. *Proc. 27th SIGCSE Tech. Symp., SIGCSE Bul. 28*,1 (March, 1996).

[8] Hsu, S.: HWSAM: A Web-based Automated Homework Submission System, *FIE'98*, (November, 1998).

[9] Isaacson, P. C., and T. A. Scott,: Automating the Execution of Student Programs. *SIGCSE Bulletin 21*(2) (June, 1989).

[10] Jackson, D. and Usher, M.: Grading Student Programs using ASSYST. *Proc. 28th SIGCSE Tech. Symp., SIGCSE Bul. 29*,1 (March, 1997).

[11] Kashy, E., Thoennessen, M., Tsai, Y., Davis, N.E. and Wolfe, S. L.: Using Networked Tools to Enhance Student Success Rates in Large Classes, *FIE'97*, (Nov., 1997).

[12] Kay, D.G.: Don't Give Grades Without It: A Comprehensive Automated Grading Assistant For Student Programs, *Proc. 24th SIGCSE Tech. Symp., SIGCSE Bul. 25*,1 (1993).

[13] *Mansouri, F. Z., Gibbon, C.A., and Higgins C.A.: PRAM: prolog automatic marker ITiCSE'98* (Aug., 1998).

[14] McGill, Tanya and Hobbs, Valerie: A Supplementary package for distance education students studying introductory programming. *Proc. 27th SIGCSE Tech. Symp., SIGCSE Bul. 28*,1 (March, 1996).

[15] Medley, D.M.: On-line finals for CS1 and CS2. *ITiCSE'98* (August, 1998).

[16] Merat, F. L. and Chung, D.: World Wide Web Approach to Teaching Microprocessors, *FIE'97*, (Nov., 1997).

[17] Naccache, H., Lindquist, T. and Urban, S.: A Framework and Reusable Tools for Developing Interactive Course Web Sites *FIE'98*, (November, 1998).

[18] Pilgrim, C.J. and Leung, Y.K.: Appropriate use of the Internet in Computer Science courses. *ITiCSE'96.* (June, 1996).

[19] Price, R, Ruehr, F., and Salter, R.: Web-based Laboratories in the Introductory Curriculum Enhance Formal Methods. *Proc. 27th SIGCSE Tech. Symp., SIGCSE Bul. 28*,1 (March, 1996).

[20] Reek, K. A.: The TRY System, or How to Avoid Testing Student Programs," *SIGCSE Bulletin 21*(1) (Feb., 1989)

[21] Roantree M. and Keyes, T.E.: Automated collection of coursework using the Web. *ITiCSE'98* (August, 1998).

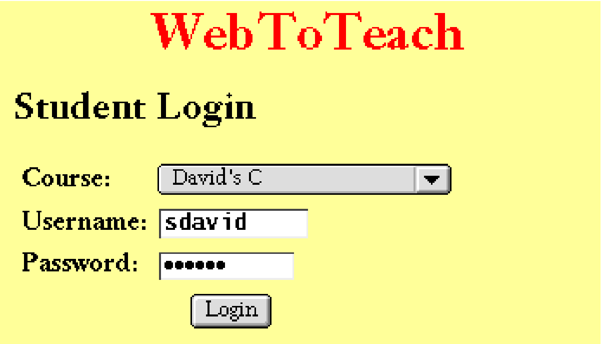
[22] Rodger, Susan H., and Walker, E.L.: Activities to Attract High School Girls To Computer Science. *Proc. 27th SIGCSE Tech. Symp., SIGCSE Bul. 28*,1 (March, 1996).

[23] Schorsch, Thomas M.: CAP: An Automated Self-assessment Tool to Check Pascal Programs for Syntax, Logic and Style Errors *Proc. 26h SIGCSE Tech. Symp., SIGCSE Bul. 28*,1 (March, 1995).

APPENDIX I: A STUDENT SESSION

This appendix illustrates a session showing student use of WebToTeach.

The student accesses WebToTeach and selects a course and enters account information:



WebToTeach

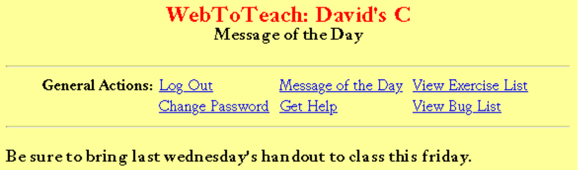
Student Login

Course: David's C

Username: sdavid

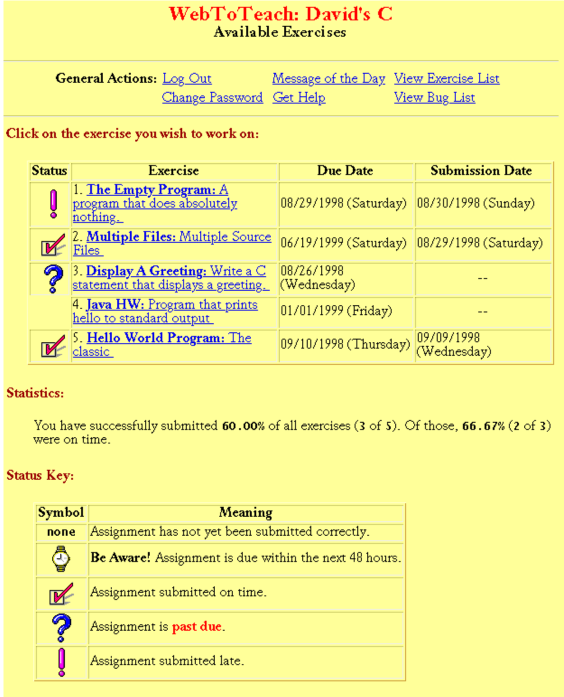
Password: ••••••

After clicking Login, the message of the day and general menu is displayed:


WebToTeach: David's C
Message of the Day

Be sure to bring last wednesday's handout to class this friday.

The student clicks "View Exercise List" and list of exercises and the student's status is displayed:


WebToTeach: David's C
Available Exercises

Click on the exercise you wish to work on:

Status	Exercise	Due Date	Submission Date
!	1. **The Empty Program:** A program that does absolutely nothing.	08/29/1998 (Saturday)	08/30/1998 (Sunday)
☑	2. **Multiple Files:** Multiple Source Files	06/19/1999 (Saturday)	08/29/1998 (Saturday)
?	3. **Display A Greeting:** Write a C statement that displays a greeting.	08/26/1998 (Wednesday)	--
	4. **Java HW:** Program that prints hello to standard output	01/01/1999 (Friday)	--
☑	5. **Hello World Program:** The classic	09/10/1998 (Thursday)	09/09/1998 (Wednesday)

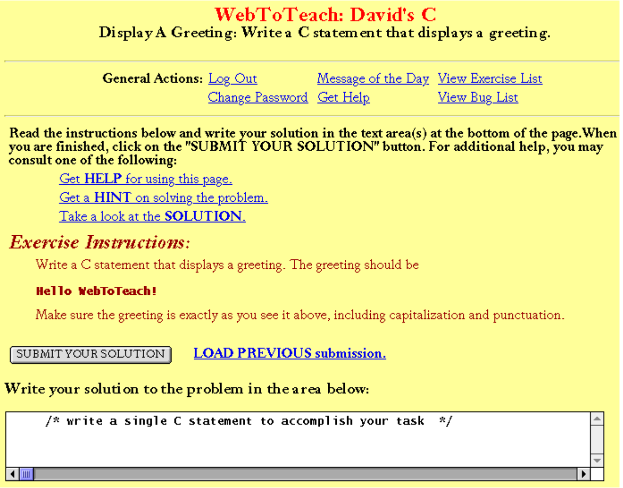
Statistics:

You have successfully submitted **60.00%** of all exercises (**3** of **5**). Of those, **66.67%** (**2** of **3**) were on time.

Status Key:

Symbol	Meaning
none	Assignment has not yet been submitted correctly.
⌚	**Be Aware!** Assignment is due within the next 48 hours.
☑	Assignment submitted on time.
?	Assignment is **past due**.
!	Assignment submitted late.

The student clicks on the third exercise and the following screen results:


WebToTeach: David's C
Display A Greeting: Write a C statement that displays a greeting.

Read the instructions below and write your solution in the text area(s) at the bottom of the page.When you are finished, click on the "SUBMIT YOUR SOLUTION" button. For additional help, you may consult one of the following:

Get **HELP** for using this page.
Get a **HINT** on solving the problem.
Take a look at the **SOLUTION**.

Exercise Instructions:

Write a C statement that displays a greeting. The greeting should be

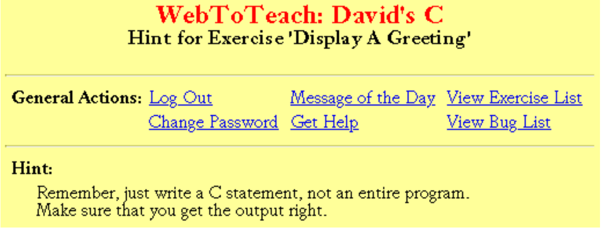
Hello WebToTeach!

Make sure the greeting is exactly as you see it above, including capitalization and punctuation.

Write your solution to the problem in the area below:

```
   /* write a single C statement to accomplish your task  */
```

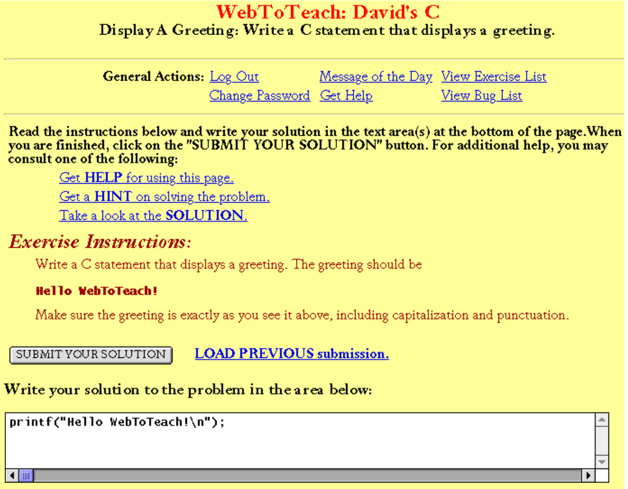
The student wants a hint and clicks "Get a HINT":


WebToTeach: David's C
Hint for Exercise 'Display A Greeting'

Hint:

Remember, just write a C statement, not an entire program.
Make sure that you get the output right.

Pressing the BACK button, the student returns to the exercise screen and types a solution:


WebToTeach: David's C
Display A Greeting: Write a C statement that displays a greeting.

Read the instructions below and write your solution in the text area(s) at the bottom of the page.When you are finished, click on the "SUBMIT YOUR SOLUTION" button. For additional help, you may consult one of the following:

Get **HELP** for using this page.
Get a **HINT** on solving the problem.
Take a look at the **SOLUTION**.

Exercise Instructions:

Write a C statement that displays a greeting. The greeting should be

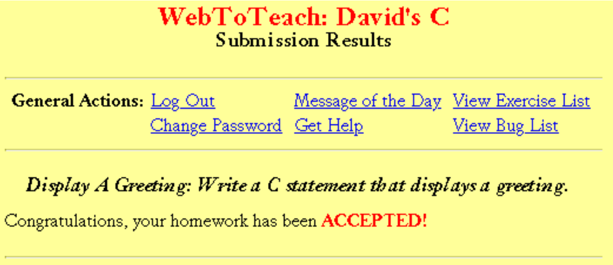
Hello WebToTeach!

Make sure the greeting is exactly as you see it above, including capitalization and punctuation.

Write your solution to the problem in the area below:

```
printf("Hello WebToTeach!\n");
```

The student clicks the SUBMIT button and within seconds gets the happy news:

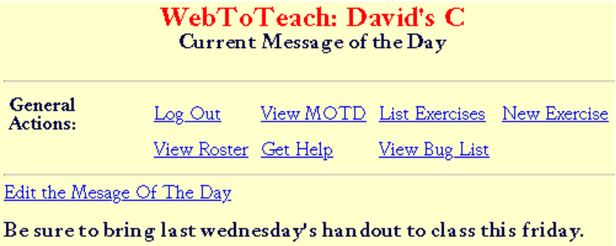

WebToTeach: David's C
Submission Results

Display A Greeting: Write a C statement that displays a greeting.

Congratulations, your homework has been ACCEPTED!

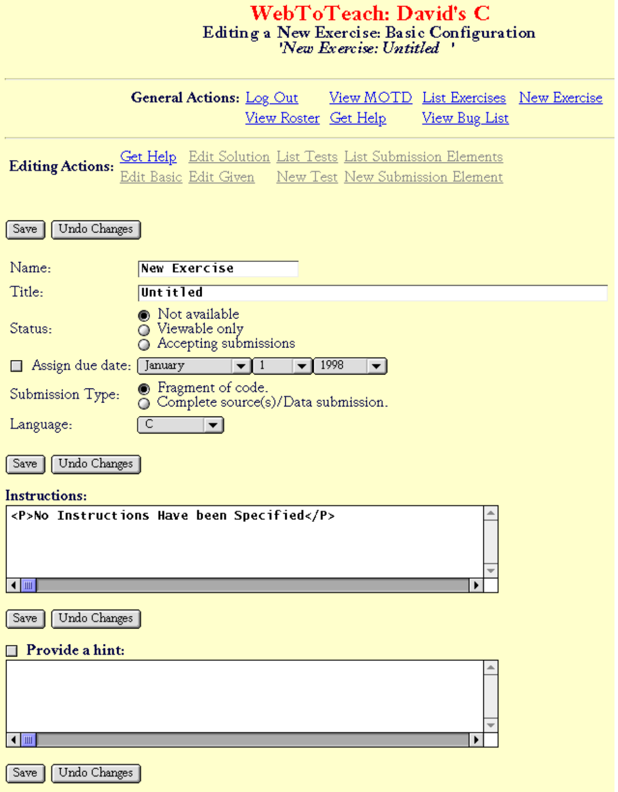
APPENDIX II: A FACULTY SESSION

This appendix illustrates a sample faculty session. Login works the same way as for students. After logging in the current message of the day for the course is displayed, along with a general actions menu:


WebToTeach: David's C
Current Message of the Day

Be sure to bring last wednesday's handout to class this friday.

To create a new exercise, the instructor clicks New Exercise:


General Actions: Log Out View MOTD List Exercises New Exercise
View Roster Get Help View Bug List
Editing Actions: Get Help Edit Solution List Tests List Submission Elements
Edit Basic Edit Given New Test New Submission Element
Save Undo Changes

WebToTeach: David's C
Editing a New Exercise: Basic Configuration
'New Exercise: Untitled '

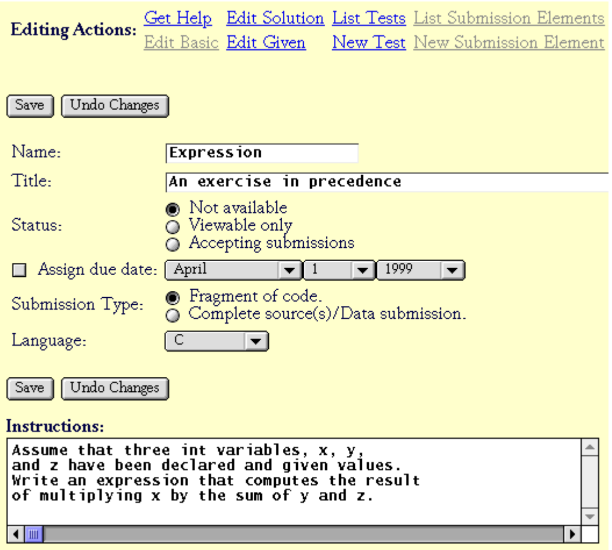
Name: New Exercise
Title: Untitled
Status: ● Not available ○ Viewable only ○ Accepting submissions
□ Assign due date: January 1 1998
Submission Type: ● Fragment of code. ○ Complete source(s)/Data submission.
Language: C

Instructions:

```
<P>No Instructions Have been Specified</P>
```

□ Provide a hint:

After filling in the exercise name, title and instructions fields, possibly electing to provide a hint and setting a due date and selecting a language, the instructor clicks the SAVE button. The screen is redrawn, but with NewTest (and other Editing Actions) enabled:

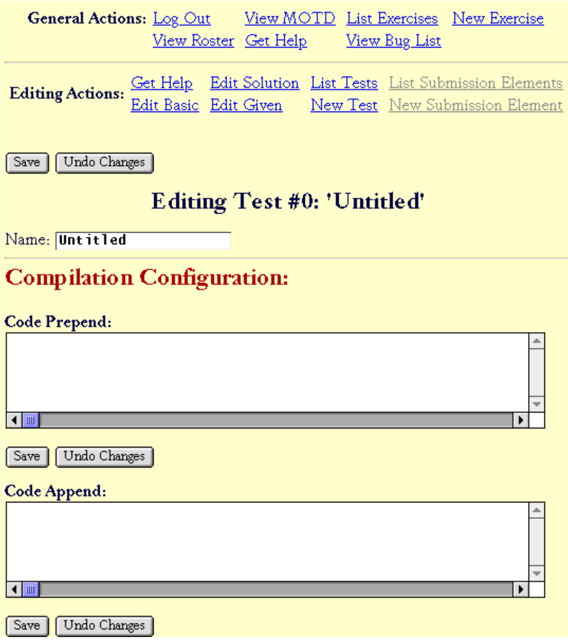

Editing Actions: Get Help Edit Solution List Tests List Submission Elements
Edit Basic Edit Given New Test New Submission Element
Save Undo Changes

Name: Expression
Title: An exercise in precedence
Status: ● Not available ○ Viewable only ○ Accepting submissions
□ Assign due date: April 1 1999
Submission Type: ● Fragment of code. ○ Complete source(s)/Data submission.
Language: C

Instructions:

```
Assume that three int variables, x, y,
and z have been declared and given values.
Write an expression that computes the result
of multiplying x by the sum of y and z.
```

Clicking NewTest allows the instructor to define a test for this exercise


General Actions: Log Out View MOTD List Exercises New Exercise
View Roster Get Help View Bug List
Editing Actions: Get Help Edit Solution List Tests List Submission Elements
Edit Basic Edit Given New Test New Submission Element
Save Undo Changes

Editing Test #0: 'Untitled'

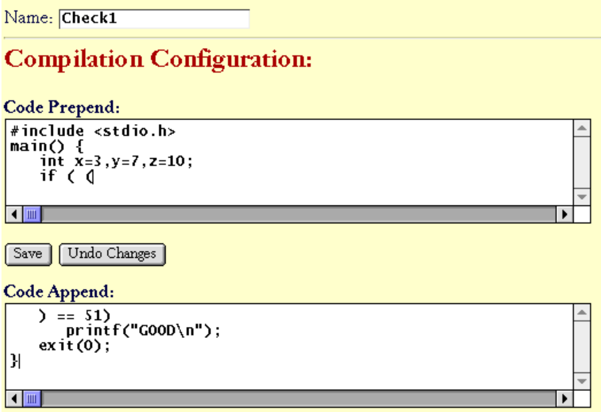
Name: Untitled

Compilation Configuration:

Code Prepend:

Code Append:

This is the key step for faculty. We must fill in the Prepend and Append boxes with code that will surround the students expression and produce output that we can check:



Name: Check1

Compilation Configuration:

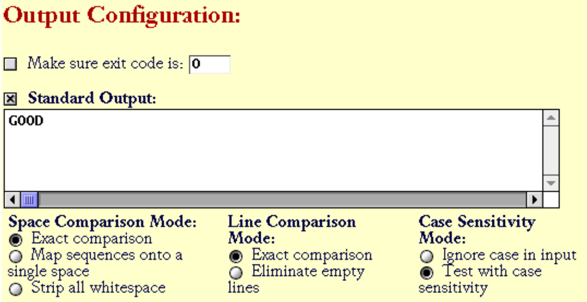
Code Prepend:

```
#include <stdio.h>
main() {
    int x=3,y=7,z=10;
    if ( (
```

Code Append:

```
    ) == 51)
      printf("GOOD\n");
    exit(0);
}
```

Elsewhere on the form, we specify a test for expected standard output:



Output Configuration:

□ Make sure exit code is: 0

☒ Standard Output:

```
GOOD
```

Space Comparison Mode: ● Exact comparison ○ Map sequences onto a single space ○ Strip all whitespace

Line Comparison Mode: ● Exact comparison ○ Eliminate empty lines

Case Sensitivity Mode: ○ Ignore case in input ● Test with case sensitivity

For each detectable point of failure (compilation, linking, output comparison, exit code, etc.) a custom, faculty-provided hint for the student can be provided.

There are a good number of other options as well, but the above illustrates the essence of the system.