Abstract

***MORTON, JEREMY ANDREW*. Advancement of Online Systems in Engineering by Expert TA. (Under the direction of Dr. Larry Silverberg.)**

This dissertation introduces a new online system called Expert TA. The system was developed based on the hypothesis that expressions are key elements in engineering problems and that the treatment of expressions is critical to the advancement of online systems. This dissertation identifies ergonomic problems with expression entry that Expert TA overcomes through the use of a problem-customize integrated expression editor, called a palate. Then the dissertation shows, using an expression analyzer that operates in the background of Expert TA, that specific mathematical mistakes within an entered expression can now be located. Emulating standard instructional practices, detailed feedback pertaining to specific mistakes and grading on the basis of specific mistakes is now possible.

Advancement of Online Systems in Engineering by Expert TA

by

Jeremy Morton

A dissertation submitted to the faculty of
North Carolina State University
in partial fulfillment of the
requirement in the Degree of
Doctorate of Philosophy

MECHANICAL AND AEROSPACE ENGINEERING

Raleigh, NC

APPROVED BY:

_____ _____

_____ _____

_____

Chair of Advisory Committee

Biography

Jeremy Morton was born to Eddie and Marilyn Morton in Charlotte, NC in 1976 and has taken great joy in mathematics throughout his life. He discovered a love for teaching during his graduate degrees at North Carolina State University. Jeremy taught every semester during his PhD, lecturing large sections of dynamics, statics, and fluid mechanics, and was nominated twice for university wide teaching awards. The statement "I dedicate my professional knowledge and skill to the advancement and betterment of human welfare" is the first statement in the official *Engineers' Creed* adopted by the National Society of Professional Engineers. Jeremy most enjoys mathematics but his desire to use mathematics for the betterment of human welfare is what makes him an engineer at heart.

Acknowledgements

I would first of all like to thank my advisor Dr. Larry Silverberg. Dr. Silverberg was my dynamics instructor many years ago when I was only an undergraduate student, but even then I knew that he was going have a considerable impact on my life. The following semester, under his advisement, I did an independent study related to his research on MRI and we have worked together since. Aside from the obvious gratitude for the knowledge and judgment that he has passed to me I would simply like to thank him for his friendship.

I would like to thank my parents for all that they have done for me. They have been incredibly supportive of me throughout my life and I have no doubt that they will remain to be so. I would also like to thank my loving girlfriend Zuzana who has stood by me throughout my entire PhD candidacy.

Concerning this research I would most like to recognize the tireless work and dedication of Robert Oliver, my partner in this project. After the mathematical structure and foundation of algorithmic partial credit grading were laid out, I recruited Robb to guide in the implementation of a web-based system. Robb is a professional programmer in RTP whose expertise and creativity have been essential to the development of this system. Robb is quite simply one of the most talented problem solvers that I have ever had the pleasure of working with.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Asynchronous, online instruction is the most popular form of online instruction [1, 2]. These are the online lectures, tutorials and the online graders that students use when they are studying and doing homework. Among their attributes, students can access material at any time and anywhere with internet access, instructors can upload material and keep it current, and the material can be dynamic and hence respond to a student's needs. Significant improvements in learning have been reported with the use of these systems [3-7].

While the above is encouraging for online instruction as a whole, it doesn't apply specifically to engineering. Presently, online systems are being used routinely in lower-level college physics, math and chemistry classes. In contrast, the adoption of online systems in engineering has been slow. In *The Andes Tutoring System: Lessons Learned* it was stated "We believe the lack of acceptance is due in part to their tendency to require extensive modification of the course content" [8]. This "…difficulty for the teacher [in] modifying his/her pedagogic strategies according to the student use of web self-learning tools" was recognized by Michau, Gentil, and Barrault [9]. When implementing an online system in an engineering class at the University of Oklahoma, Gramol stated, "…. all homework, quizzes, and tests were also designed so that they could be delivered over the Internet. This required all questions to be either fill-in-the-blank or multiple-choice which is a major change in the normal teaching style where partial credit is awarded for wrong problems. This was the single most difficult change that the author had to do in implementing an all-electronic course. Traditionally, partial credit is given to students in engineering courses due to the difficulty in solving problems and frequency of simple errors such as algebra and sign errors." [10]. In another article the question was posed, "Why has undergraduate engineering education lagged behind some other fields in adopting online methodologies?" Their answer was that "… the special needs of undergraduate engineering education have not been well served by [existing] methods of online education."[11]. As stated in [11], the problem stems from the simple fact that in engineering instruction, the "…significant use of mathematics [has] not been as easy to implement." "Moreover, the main drawbacks to e-learning are linked to the poor cognitive capacities of computer systems. At the present time, they are far from being

adaptive to the user."[9]. It would seem that the op-scan of the 1970s would have been widely adopted in engineering had expressions not been critical. It is unlikely that the solution to this problem can be achieved by transforming engineering instruction to match the existing limited functionality of online systems. It is more likely that online systems will adapt to meet the instructional needs of engineering classes. Thus, it is first hypothesized that: ***The treatment of expressions is critical to the adoption of online systems in engineering classes.***

Chapter 2 reviews the current state-of-the-art of online systems. It turns out that certain human factors need to be solved before pedagogy can be considered. Is it even feasible for students to enter expressions into online systems, and if not, can the problems with entering expressions be identified and overcome? The review first focuses on the problem of expression entry in online systems and then pedagogy is considered. Harrington and Reasons of the University of Southern Indiana recommend to any department or college considering an online system to ask the "… fundamental questions: Why are we doing this in the first place? ..." [12]. Stated a little differently, but essentially the same, the relationship between accepted practices and the capabilities of an online system needs to be considered. The question must be asked whether and to what extent existing online systems meet instructional needs. Accepted practices use tried and true methods and hence represent an experiential base from which to draw. Through a review of accepted practices, major areas in which online systems are potentially effective can be identified. In particular, one can follow the treatment of expressions throughout the instructor's process and gain a better understanding of how expressions are treated and hence how an online system might treat them. Thus, *Chapter 3* reviews accepted practices in engineering and formulates a problem statement for the advancement of online systems. *Chapter 4* presents the solution provided by Expert TA. Expert TA operates with an expression analyzer in the background. Using the expression analyzer, each mistake that a student makes within an expression is found. Detailed feedback is provided and grading is performed on the basis if specific mistakes that a student makes. Chapter 5 is a review of the pedagogical facets of this research and Chapter 6 offers a deeper look into the mathematics that powers the system.

Chapter 2. The State-of-the-Art

First consider expression entry. Expression entry is a process performed by a student to transform an expression into electronic form. The process competes to some degree with the process of writing down an expression on a sheet of paper. In expression input (and text input) complex writing motions of the hand are replaced with striking keys and clicking a mouse. If the set of keys on the keyboard is complete, expression entry can be comparably efficient to writing an expression on a sheet of paper, not to mention the advantages that exist once an expression is in electronic form. Unfortunately, the standard keyboard is not complete. It is designed for text entry, not expression entry. The standard keyboard does not have Greek letters or mathematical operators, so expression input is difficult without a special tool. The expression entry problem is addressed by the expression editor (also called an equation editor). The expression editor is a complimentary tool comprised of modules organized by mathematics similarity for expression entry. The expression editor provides complete capabilities in formatting and functions (symbols and operators) and is designed for efficient navigation.

The act of entering an expression requires time and special knowledge. Special knowledge refers to knowledge that is specific to the expression editor. The student must become familiar with the expression editor and learn how to navigate quickly and perform operations properly. Time is almost always a consideration when an expression editor is used, and special knowledge is almost always a consideration when one is not used. A number of studies have been performed that measure a user's computer abilities at the beginning of a semester. These studies show that student success correlates directly with his or her computer abilities [5]. These findings are consistent with the problem discussed in this dissertation. Students who are not proficient with an expression editor or who have had little exposure to math programming syntax are not expected to perform as well when using on line systems than students who are proficient with an expression editor and who have had more exposure to math programming syntax.

Hauk and Segalla performed a study on student perceptions of the online homework program called WebWork. WebWork was used in twelve moderately sized college algebra classes [15]. With regard to student entry, that paper found that "The frequency with which students remarked on the difficulties of inputting their solutions in the notational language common to graphing calculators and computer algebra systems was a concern. WeBWorK version 2, release 3, attempts to address this concern with drop down function menus containing commonly used macros like sqrt[ ] to indicate square root and abs[ ] for absolute value."[14]. Their work reinforces the viewpoint that expression entry requires special attention. Simply supplying the student with any mechanism to input an expression is not sufficient. ***The problem of student input of expressions is an optimization problem in which the time to input an expression and the special knowledge required on the part of a student are minimized.***

2.1 General Syntax Help

Figure 1 shows an expression input prompt in the commercial system called WebAssign [15]. WebAssign does not use an expression editor. Expressions are entered using syntax, similar to that used in graphing calculators and math programming packages.

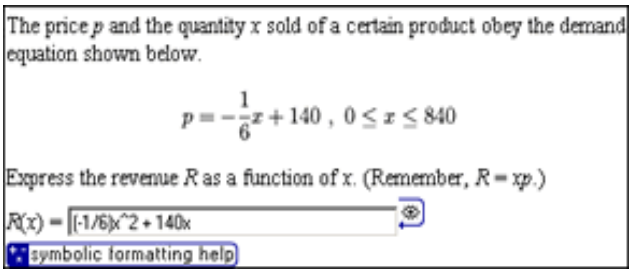


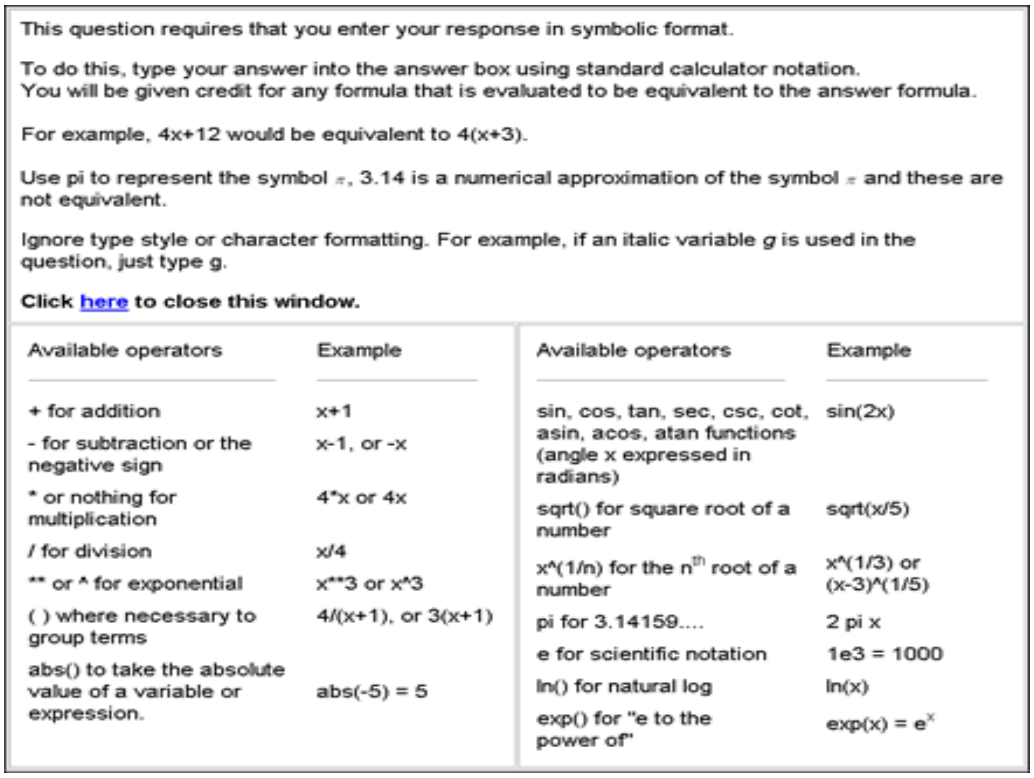
Figure 1: Example of a help button for syntax help

As shown, there is a button at the bottom of the problem to help with syntax. By clicking this button, a separate help window opens with general syntax help.

Figure 2 shows the help window. As shown, there are about a dozen tips to guide the student. In some cases the displayed help answers the student's question, and in other cases it doesn't.

Every question can't be answered unless a comprehensive help window is offered but that would overwhelm the user.

Figure 1 displays a function of x. If a function of θ needs to be entered, then another syntax problem arises because there is no θ key on the keyboard. In some computing environments, such as when programming in HTML, the variable θ is written as θ and in other environment it's simply written as theta. Even a simple issue as this is not dealt with in the help window, again, because it is impractical to address every syntax question.



This question requires that you enter your response in symbolic format.

To do this, type your answer into the answer box using standard calculator notation.
You will be given credit for any formula that is evaluated to be equivalent to the answer formula.

For example, 4x+12 would be equivalent to 4(x+3).

Use pi to represent the symbol π, 3.14 is a numerical approximation of the symbol π and these are not equivalent.

Ignore type style or character formatting. For example, if an italic variable *g* is used in the question, just type g.

Click here to close this window.

Available operators	Example
+ for addition	x+1
- for subtraction or the negative sign	x-1, or -x
* or nothing for multiplication	4*x or 4x
/ for division	x/4
** or ^ for exponential	x**3 or x^3
( ) where necessary to group terms	4/(x+1), or 3(x+1)
abs() to take the absolute value of a variable or expression.	abs(-5) = 5

Available operators	Example
sin, cos, tan, sec, csc, cot, asin, acos, atan functions (angle x expressed in radians)	sin(2x)
sqrt() for square root of a number	sqrt(x/5)
x^(1/n) for the n^{th} root of a number	x^(1/3) or (x-3)^(1/5)
pi for 3.14159....	2 pi x
e for scientific notation	1e3 = 1000
ln() for natural log	ln(x)
exp() for "e to the power of"	exp(x) = e^x

Figure 2: Example of syntax help

When a student is required to know information that is specialized to expression input, instruction is being compromised. The student is now being evaluated on his or her knowledge of the subject as well as on his or her special knowledge of the syntax used by a particular online system. The general help menu leaves student syntax questions unanswered or results in an overwhelming amount of information – both problems call into question the efficacy of general help menus.

2.2. Problem-Customized Syntax Help

A more advanced approach to supplying syntax help is to customize it to a problem. In *Web-Based Agents for Reengineering Engineering Education* an educational technique focuses on the use of an "… adaptive lecture guide that provides navigation guidance sensitive to students knowledge status" [3]. A similar strategy can be employed when providing help to students concerning syntax issues. If the help contains only information relevant to the current problem, it can be complete without being overwhelming. The Andes Physics Workbench was developed at the University of Pittsburg and the US Naval Academy [16]. The system does not use an expression editor approach but rather requires an expression to be entered in notation similar to that used with graphics calculators and provides customized syntax help. As the system was being developed they "… tried many traditional user interface training methods: tutorials, manuals, a Microsoft style help system, an animated talking agent and unsolicited pop-up hypertext. None worked particularly well."[16] The troubles they encountered reinforce the viewpoint that simply supplying a student with some mechanism to input an expression does not suffice. While, their technique for customized syntax help overcomes problems with syntax help, ultimately it does not overcome the difficulties of expression entry. "Several instructors who were considering Andes for use in their courses were able to start solving problems without any user interface training at all. This is a tribute to its intuitive, paper-like interface. However, they soon ran into user interface details that stymied them, and rejected Andes as '…too hard for students to use...' The underlying problem is that Andes requires mathematical and graphical notation to be used precisely." [16]. Perhaps the single most important lesson learned is that an expression editor of some kind is needed.

2.3 The Expression Editor

Expression editors are segmented into modules of similar operators, functions, variables and separators. To reach a module, the user follows a route to the module. The navigation time is associated with following the routes. A fully-functioning expression editor has a large number of modules and a correspondingly large navigation time. Indeed, navigation time is proportional to functionality. In a typical expression editor, the navigation time to enter an expression is considerably larger than the time required to write an expression by hand or to enter an expression electronically in a rougher form. The navigation time increases

considerably when a user is not familiar with the routes to the modules and the syntax being followed. To become familiar with the routes to the modules and the syntax, a learning curve exists. After some period of time the user will have special knowledge of the routes and the syntax needed. The time to reach this special knowledge is proportional to the expression editor's level of functionality. It is reported that students spend significantly more time on homework assignments with online systems than with pen and paper [5]. Students spending more time on homework can certainly be viewed in a positive light when the time spent is educational. However, the additional navigation time is clearly a drawback. Fully-functioning expression editors are invaluable when presentation is a priority and time is not a factor. When time is a factor, and presentation is not a concern, the use of a fully-functional expression editor is superfluous. In online instruction, expression content is the primary concern, not presentation.

2.4 The Customized Expression Editor

A technique employed in some online systems is to provide an expression editor, but to limit its functionality to functions needed in a particular set of problems. The functionality is sufficiently limited to enable most or all of the modules to be accessed directly from one screen. Navigation is confined to finding modules on one screen so navigation time is reduced. A number of commercial online evaluation systems use this technique, such as WebCT. WebCT gives students the option to enter expressions using "... a customized version of Design Science's WebEQ® technology ..." (See Fig. 3) [21,17].

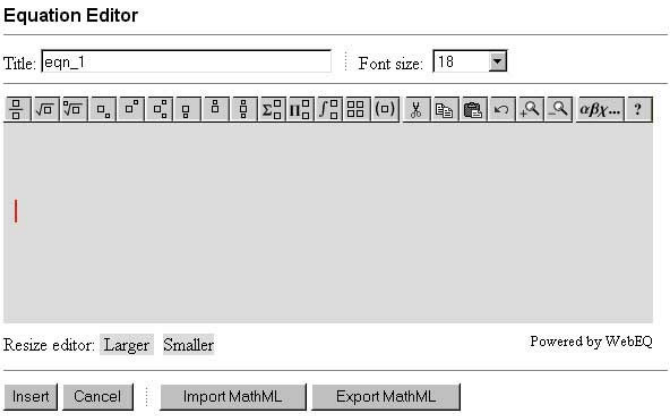


Figure 3: The problem-customized expression editor

As shown, buttons that address formatting and symbols are arranged in a row at the top of the screen. Syntax problems are removed using this interface. When a student wants to use the square root function he or she simply clicks on the button with that function and the navigation to the symbol for $\sqrt{\ }$ is relatively quick. The company Brownstone offers a system called Ph. Grade-Assist through the publisher Prentice Hall. Ph. Grade-Assist also offers a customized version of WebEQ that is similar to that which is shown in Fig. 3. This tool is relatively concise and offers everything that a student would need for a given problem. Figure 4 shows the window that appears for the user if the trig function button is selected in Ph Grade-Assist [21,18].

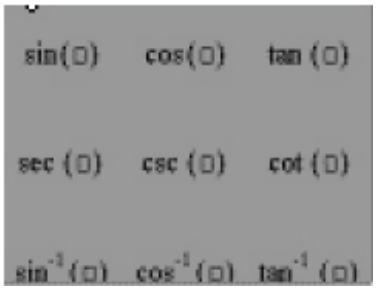


Figure 4: Example of a trig function window

The student can input a trig function without raising any syntax questions. This approach reduces both syntax problems and navigation time but not necessarily to a sufficient degree. Navigation time is still a factor. For example, to enter an expression that contains several trigonometric functions and Greek symbols, like sin(θ)cos(θ), the student must navigate considerably. He must navigate to the trig module, select sin( ), navigate to the Greek symbol module and select θ , navigate again to the trig module, select cos( ), and navigate again to the Greek symbol module and select θ. As another example, to enter an expression that contains variables not contained in the editor, like $E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2 + mgh,$ the student switches between the mouse and the keyboard repeatedly. The back and forth motion between the mouse and the keyboard to enter the variables m, k, x, g, and h, and the numbers in the expression is much slower than typing. Overall, the back and forth motion between the expression editor, the mouse, and the keyboard is a complaint among students. The problem-customized expression editor reduces special knowledge and navigation time of the general expression editor, but typing in expressions using the problem-customized expression editor

is still time consuming and tedious. This completes the review of the main problems of expression entry in the present state-of-the-art. The solution is presented by Expert TA later in the dissertation.

2.5 Tutoring

Next, consider the state-of-the-art of tutoring in online systems. A representative set of state-of-the-art online systems consists of WebWork, WebAssign, WebCT, Ph Grade Assist, AIM, Andes, WebMentor, Brainbench, Virtual U, and Mastering Physics [14, 15, 17 – 24]. These systems offer the student hints when they're working on a problem. The treatment of hints in Mastering Physics is particularly sophisticated [24]. The most basic systems offer problems divided into steps, and questions of the form multiple choice, true or false, and multiple select. Very few dynamic hints or feedback are offered except whether an answer is correct or not. The hints represent static content and hence are technically easy to provide online. The dynamic content being offered indicates whether an answer is correct or not. Typical feedback is "The answer is wrong.", "The answer has the wrong number of significant figures." and "The given units are inconsistent." The systems do not provide feedback that analyzes expressions in any detail. The online systems evaluate expressions in a binary, right or wrong, process using expression evaluators. Essentially an expression evaluator replaces every occurrence of a variable in an expression with an arbitrary number and the, now numeric, expression is calculated resulting in a number. This number is compared to the number generated from the correct expression to determine if the expression is correct. Specific feedback can't be generated because the resolution is lost in the numeric compression. AIM and Mastering Physics are two systems that offer a level of non-binary feedback. AIM was developed to target higher level classes. It's assessed in "Assessing Higher Mathematical Skills Using Computer Algebra Marking through AIM" [19]. In AIM, the student can be asked to enter an integral of a function and AIM differentiates the entry so the student can compare it with the original function. In Mastering Physics, searches are performed in the background to inform a student whether he or she has used the correct variables in an expression.

In general, asynchronous online systems are able to reproduce the routine tasks found in engineering education. The tasks that involve higher-level cognition tend to be more difficult to treat online. For example, placing lectures online and providing general hints to problems

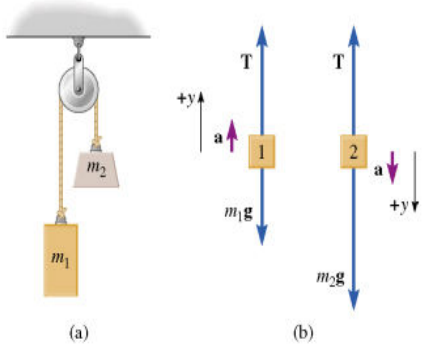
online are relatively simple tasks. However, providing detailed feedback in response to specific mistakes made by a student is much more difficult. The capability of treating expressions in detail, which is critical to engineering education, is not found in commercially availably online systems. The solution to this problem is provided by Expert TA as described later in the dissertation.

2.6 Assessment

Three widely-used commercially available systems that offer engineering content are WebCT, WebAssign, and Ph Grade Assist. The problems contained in these systems are predominantly multiple choice, true-false, and short response. When expressions are involved, they're graded as right or wrong. Figure 5 shows a problem from the WebAssign system. As shown, the student is asked to determine the time required for block 2 to reach the floor. Short response questions, such as this, can only be graded as right or wrong. The numerical value represents a low resolution view of the student's work, given that no information pertaining to a student's mistakes can be extracted from the numerical value. In addition most engineering problems, such as this one, require several pages of work to reach the final answer.

2. [GRR2 4.P.012.] In the figure below (part a), two blocks are connected by a lightweight, flexible cord which passes over a frictionless pulley. If m_1 is 3.5 kg and m_2 is 9.3 kg, and block 2 is initially at rest 150 cm above the floor, how long does it take block 2 to reach the floor?

s



Submit this question only | Save work

Figure 5: A Problem in the WebAssign System

The WebAssign system [15] is offered with four engineering text books: Engineering Statics and Engineering Dynamics [25,26], Mechanics of Materials [27], and Fluid Mechanics [28]. On average, 230 problems are available online from each textbook. Less than 10% of these problems involve expressions. The other systems similarly offer a small percentage of problems involving expressions.

Regarding the problem of assessment, Shapiro notes "When a student enters an incorrect expression, the tutor needs to try to identify what is wrong. As we do not know what expression the student was aiming at, a comparison of expressions may not be useful. One way to find the source of some errors is to perturb the entered expression in various ways, and ask the algebra subsystem if the resulting expression is correct." [29] He continues, stating that the "…large number of perturbations can be checked. Nonetheless, many near misses will not be identified by this process." As Shapiro states, the attempt to randomly perturb an expression to the correct form does not solve the problem. Numerical expression evaluators destroy the resolution of an expression and are not suitable tools for finding mathematical mistakes. To solve the problem at hand an organized way of symbolically deconstructing expressions is needed. The specific challenge is associated with properly analyzing open-form expressions in a manner that is conducive to identifying mistakes and offering partial credit. Presently, no system is capable of finding a student's mistakes in an expression that has been entered, and hence no system assesses student work based on her or his specific mistakes. The solution to this problem is provided by Expert TA as described later in the dissertation.

Chapter 3. Standard Practices in Instruction

Instruction is a structured process in which educational information is conveyed by an instructor to a student. In engineering, the student learns about the methods and practice of problem-solving. New information is conveyed most frequently through readings and lectures. Information is then reinforced by working problems and the work is assessed through a grading rubric. In engineering classes, there is great deal of new information to convey, so students and instructors alike are aware of the time constraints imposed on the process. The major contribution of asynchronous online systems thus falls into the general categories of lectures, tutoring, and assessment (See Fig. 1). Successful online systems that supplement or replace an instructor's lecture have already been adopted in engineering, most frequently in long-distance education; this type of asynchronous online system is not being considered here. The problem at hand applies to online tutoring and assessment. The question arises where the online tutoring and assessment can be helpful to the student and the instructor.

Instructor	Student
Lecture	Online Lecture
Tutoring	Online Tutoring
Assessment	Online Assessment

Figure 6: Basic Elements of Class Instruction

3.1 Tutoring

While individual styles vary, tutoring follows a structured process. To better appreciate the process, imagine that a student stops by your office to ask a question about a problem. How do you handle the question? The instructor, when answering a student's question, follows a routine to determine the origin of the student's lack of understanding and then offers guidance. The instructor recognizes whether the student has yet attempted the problem, whether the student attempted the problem and stopped in the middle, or whether the student solved the problem and arrived at an incorrect answer. The guidance can be in the form of a

general hint that pertains to the problem or in the form of explicit feedback in response to an error that was made. To offer explicit feedback, the instructor may first review the student's attempt at the solution. Ideally, the instructor sits down with the student, watches the student solve a problem, and observes his or her problem-solving habits in order to identify the origin of the student's mistake and offer detailed feedback. By sitting down with the student, a mistake is identified as soon it's made, the student is prevented from proceeding after making a mistake, and lost time during which a student is stuck is eliminated.

As a rule, instructor does not answer every question, as much as reinforce the steps that should be followed to answer a question. Ultimately, the instructor wants the student to learn how to think through a problem himself. The least amount of information is provided that enables the student to proceed with solving a problem independently. In summary, the tutoring process provides a) guidance through hints and b) feedback in response to mistakes. The help is aimed at enabling the student work through a problem independently and in a timely manner.

The instructor commonly guides the student with general hints when the student has not yet attempted a problem, or has attempted a problem and stopped in the middle. For example, the hints that a tutor provides, from general to specific, can be of the following types:

- Problem classification and relationship to lecture and readings in the text
  Example: "Please read Section 10.2."
- Readings of similar example problems
  Example: "Please read Example 10.2 – 5. It is similar to your problem."
- Hints relating to the problem at hand
  Example: "Be careful to correctly label the direction of the friction force in the free-body diagram."
  Example: "Notice in Example 10.2 – 5 how the friction force is treated in the free-body diagram."

Specific feedback is frequently offered after a student has solved a problem and arrived at an incorrect answer. Again, the instructor begins by attempting to identify the student's mistakes. The instructor inspects the student's work. Ideally, the instructor sits down with the student and they review the work together. The instructor looks at the student's free-body diagrams, his expressions, method of solving the expressions, and ultimately his answers.

The mistakes fall into two categories: conceptual mistakes (excluding mathematical) and mathematical mistakes. Common mathematical mistakes are calculator entry errors, sign errors, algebraic errors, and trigonometric errors. Sometimes it is more tedious and time consuming to find mathematical errors than to find other conceptual errors. Whether offering guidance with hints or offering feedback in response to mistakes, time is an important factor to both the student and the instructor. When a student gets stuck, he or she looses valuable time. The student must stop working on the problem, wait for office hours, and then visit with the instructor to get the question answered before returning to the problem. Generally the student's goal, when seeking assistance from an instructor, is to have his or her question answered quickly. Going to office hours is a timely endeavor and many students avoid it unless absolutely necessary – in some cases at the expense of learning the material thoroughly. The time that an instructor needs to spend with a student can be a problem, too. It can be difficult and time consuming to evaluate a student's work in detail and locate a mathematical error.

3.2 Assessment

Assessment is largely performed by finding a student's mistakes based on a grading rubric. The grading of mistakes and comments made on graded homework are used by the student as part of his or her learning process. The student identifies where specific mistakes were made and the reasons for the mistakes so they won't be repeated in the future. Of course, the grading of mistakes is also used by the instructor. The instructor assesses student comprehension and more importantly areas of deficiency on the part of individual students and the class as a whole. The instructor uses grades, not only for assigning final grades, but to assess performance and make instructional modifications. Maintaining a high level of detail in grading is an integral part of the educational process.

Figure 7 shows a typical problem graded by an instructor. As shown, detailed mistakes are found, feedback is given that address individual mistakes, and deductions are made. At each stage of a problem, a student can make a variety of mistakes on different levels of severity. The first and most representative feature of grading is the identification of the specific place where the error is made. The second most representative feature is the provision of constructive feedback relating to the reason behind the mistake. Of course, it is difficult for

the instructor to understand the true underlying reason behind a student's mistake because multiple reasons can explain a given mistake (e.g., sloppiness, sign convention, and math deficiency). In any event, the student uses himself the location of the error and the detailed feedback to make the appropriate adjustments to his own understanding.

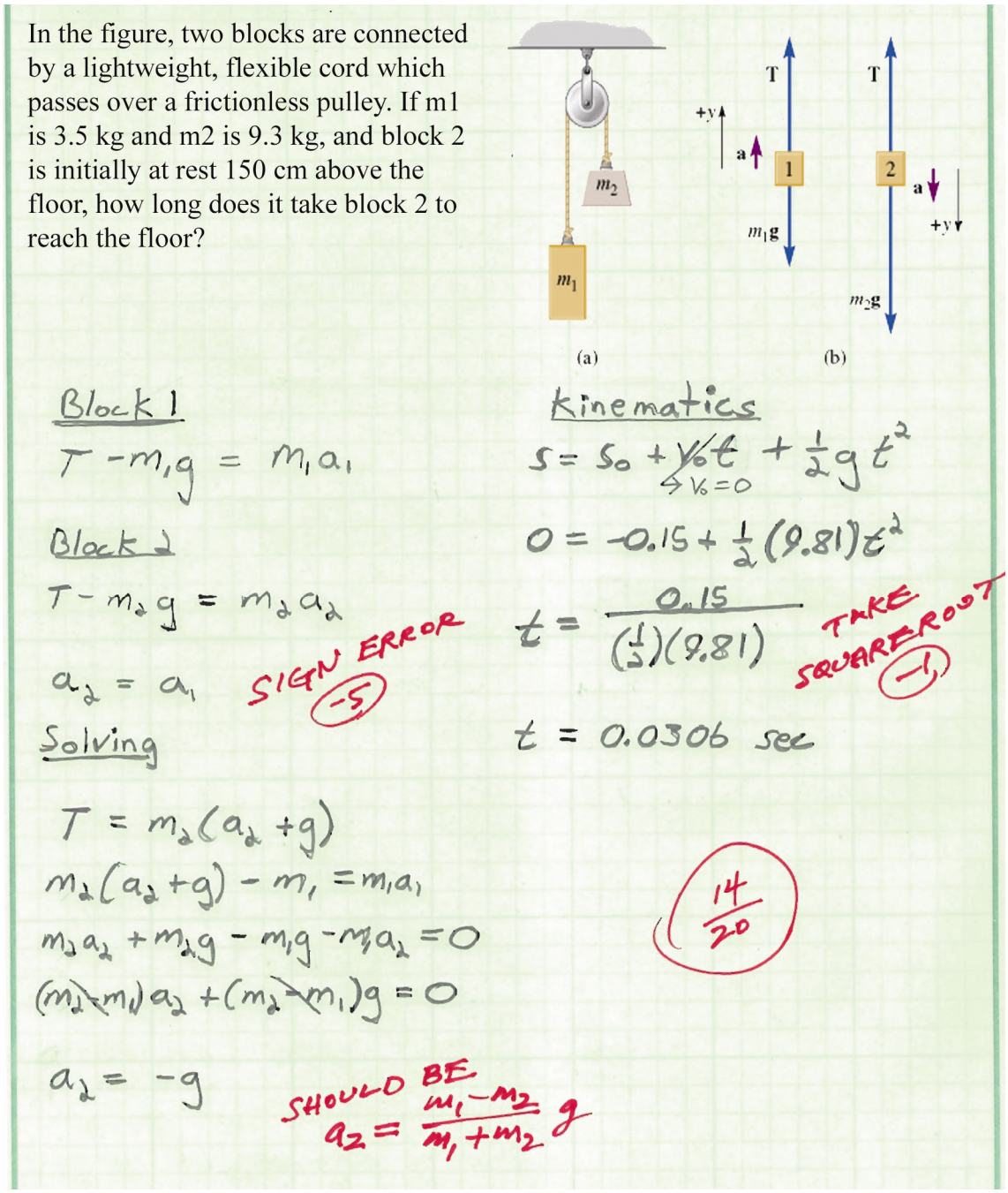


Figure 7: Grading by an instructor

A third aspect of grading that is particularly important to students is grading consistency. The complexity of the grading process makes it difficult to be consistent; perfect consistency in instructor-graded problems is impossible. Grading consistency is particularly important for the student to gain a sense of fairness. A common complaint among students arises when more points are deducted for his or her mistake than for the same mistake made by a classmate. The instructor handles the problem of grading consistency several ways. To increase resolution, he or she divides the problem into steps. For each step, the instructor anticipates types of errors and assigns a point value for each type of error. This is called a grading rubric. Some level of inconsistency is inevitable because it's impractical to anticipate every error. In summary, three important features of grading that are practiced by dedicated instructors are, in order of decreasing importance, a) finding where each mistake is made, b) providing specific feedback about each error, and c) grading consistently.

3.3 Problem Statement

In view of the preceding comments, the question arises how online systems can help students and instructors in tutoring and grading. Assume that an online system has been developed that contains a computer program that runs in the background and that is capable of analyzing an expression. The mathematical analyzer determines the mathematical errors made at a step - not just whether the expression is right or wrong but precisely what errors are made. The online tutoring system provides detailed feedback pertaining to the particular mathematical mistake. Also assume that the aforementioned problems with expression entry have been solved. Now, let's imagine that the student's problem has been divided into several steps and that the student is asked to enter a mathematical expression into the computer after completing a step.

Notwithstanding the algorithmic complexities of an expression analyzer and the design of an effective system of expression entry, the question arises how treating expressions in online system helps engineering instruction. Within the context of standard practices followed by engineering instructors, the treatment of expressions by an online system replaces much of the student's time and effort directed toward finding mathematical mistakes. Once a mathematical mistake has been identified by the online system, the student looks through his work at the given step to find the origin of the mistake. This task is much simpler and more

rapidly completed than the task of finding an error, not knowing at what step it occurred or the term in the expression that was responsible for the mistake. Furthermore, the time and effort spent on visiting with the instructor is reserved for conceptual mistakes. The instructor's time savings is also significant. The instructor generally spends time and effort on providing hints and detailed feedback. The detailed feedback pertains to conceptual (non-mathematical) mistakes and mathematical mistakes. The time and effort in finding mathematical mistakes is largely eliminated.

In general, it's time consuming to grade problems effectively. In fact, the time required to find each mistake and provide detailed feedback can be prohibitive. For example, in a class of 30 students who are assigned 10 homework problems per week, each containing 5 expressions, the instructor would need to inspect 1500 expressions per week and provide feedback for each error found. The open-form presentation of a student's work, even when neatly written, makes it difficult to inspect the expressions. The difficulty is so great that in many classes, particularly in large classes, the instructor is not able to grade thoroughly. In some cases, only a subset of the assigned problems is graded. An online system that has the ability to find the mistakes that a student makes in each expression can, based on a grading rubric, assess the student's work in detail and eliminate the time an instructor spends on grading.

Expert TA uses an expression analyzer and a palate for expression entry that requires no special knowledge and little navigation time. Thus, it helps engineering instruction in the way imagined above.

Chapter 4. The Expert TA System

Expert TA is presently a non-commercial, online tutorial and homework system that was developed for use in engineering classes and higher level mathematics and science classes. A typical problem is shown in Figure 8. The page layout is essentially the same in all problems and the layout is similar to that found in other online systems, too, e.g., Mastering Physics [24]. The problem statement is in the upper left, a figure is shown in the upper right, and the user interface for each part of a problem is displayed below. The user interface consists of a field where the expression appears, buttons to navigate between problem parts and access hints, and a customized problem-integrated expression editor used to enter expressions. Expert TA operates in two modes – homework mode (shown in Fig 8) and tutorial mode. In tutorial mode boxes for feedback and hints, which are initially empty, are located below the user interface (not shown in Fig. 8).

Expert TA is designed to emulate student-instructor interaction in office hours. The goal has been to produce an environment in which the student feels that he or she is in the presence of an instructor who can provide hints and feedback. When the student is stuck guidance is offered in the form of hints that are organized and thoughtful. Like an instructor observing a student solve a problem, detailed feedback is provided at each step. In tutorial mode, the student corrects his or her mistakes before proceeding to the next step.

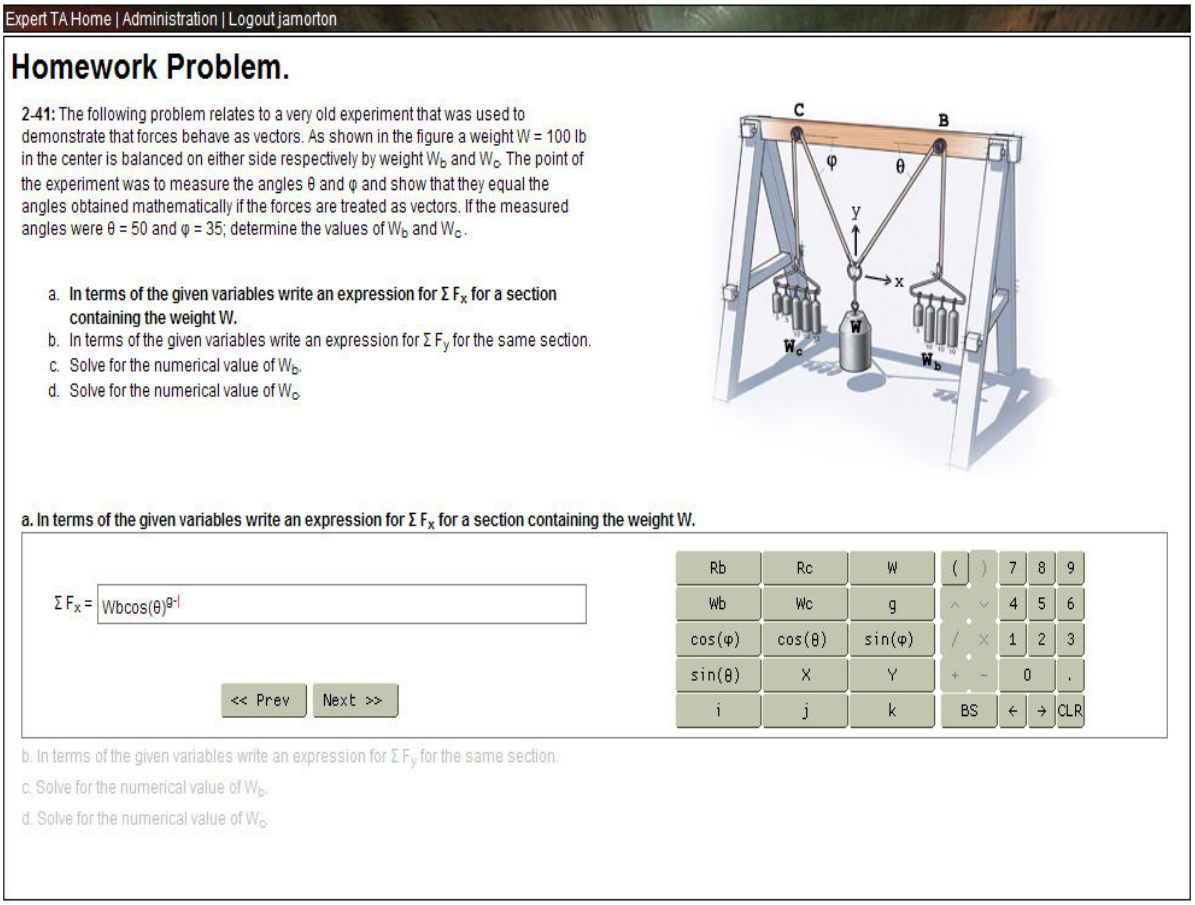


Figure 8: A typical problem

The environment is governed primarily by hints, feedback, and a mechanism through which the student enters an expression. The following discusses expression entry, the tutorial mode and the homework mode.

4.1 Expression Entry

As stated earlier, the problem of expression entry is an optimization problem in which navigation time and special knowledge are minimized. The use of a general-purpose expression editor requires a large navigation time and a large amount of special knowledge. As shown earlier problem customization reduces both syntax problems and navigation time. However, navigation time is still a problem with the problem-customized expression editor. The back and forth motion between the keyboard and the mouse is frustrating and time consuming. This problem is remedied by eliminating the back and forth motion between the keyboard and the mouse. The keyboard is integrated with the problem-customized expression

editor to produce a problem-customized integrated expression editor. The problem-customized integrated expression editor is a set of buttons that are used for expression input. It has the natural feel of a calculator, but is virtual and customized to a problem. The problem-customized integrated expression editor is called a palate. As shown, everything that the student needs to enter an expression is found in the palate. The palate is divided into two parts. The right portion of the palate, like a calculator, contains the numbers 0 through 9 and basic navigation buttons like the arrow keys, the backspace, and the clear button. The left portion of the palate contains operations specific to the problem. Additional dummy variables and symbols are included in the left portion of the palate depending on the preferences of the author of the problem, or the instructor using the problem. As shown in Fig. 8, the back and forth motion between a mouse and a keyboard has been eliminated as well as navigation, e.g., navigation to trig functions.

The look of the palate changes depending on the problem. For example, in a problem that calls for multiple-choice or true-false solutions, the palate degenerates to the appropriate form. In a problem that calls for units to be provided in the answer, the units are supplied in the buttons. The palate is a form of an expression editor that is particularly well-suited to engineering classes. Two important features of expression entry with the palate are described next.

4.2 Error Proof Expression Entry

Syntax is not nearly as limiting in hand written work (text entry) as it is in work with expressions. It is a frequent occurrence for an engineering instructor to hear a student complain about the difficulty of finding a syntax error in a computer program that he or she wrote. This is a problem in both computer programming and in online systems. The mistakes can turn out to be a “small” issue like a missing parenthesis, but when working with expressions, any mistake causes the computer program to stop resulting in a significant loss of time and the need to find a syntax error. The issue at hand is that syntax mistakes are commonly made when expressions, or computer code, are put in electronic form.

The problem of syntax error has been addressed in the Expert TA system. Referring to Fig. 8, notice in the right portion of the palate that some of the buttons are disabled and that the

quantity "$W_c \cos(\theta)^{g-}$" appears in the expression field. The quantity in the expression field represents part of a mathematical expression that is being entered. At this particular instant of expression entry, seven buttons (+, –, /, , etc.) have been temporarily disabled because, if clicked, a syntax error would result. Operator buttons in the palate are dynamically disabled to prevent mathematically invalid expressions from being entered. Even "small" mistakes, like accidentally striking the "+" button twice, are eliminated. Error proof expression entry eliminates time spent on finding syntax errors.

4.3 Keyboard Auto-Complete with Expression Entry

The palate requires the student to input an expression using the mouse. Although this approach solves the problems of navigation time and special knowledge, the student does not have the option to use the keyboard for expression input. Many students, however, feel more comfortable using a keyboard than a mouse. This problem is overcome by employing a process called keyboard auto-complete. The keyboard auto-complete process, although never applied to online systems, is an attractive process when used together with the palate.

The keyboard auto-complete process arises in other applications, for example, when searching online for airline tickets. When visiting a site like expedia.com the user is asked to enter a state of departure. As soon as the user types a "d" in the state of departure field the computer completes the "d" into "Delaware." While the user is typing, the input is algorithmically compared to valid input. The defined valid input here consists of the 50 states. In an engineering problem, the valid input consists of the palate's contents. For example, in Fig. 8, when a student types "c" on the keyboard the keyboard auto-complete function displays the valid inputs of cos(θ)and cos(θ). To narrow down the choices, the user either continues typing or selects one of them. The keyboard auto-complete process, when applied to the palate, provides a keyboard alternative to the mouse-only input required of the palate.

4.4 Guidance with Hints

Following standard practices, hints are presented from general to specific and they are offered judiciously. The goal is also to provide hints that encourage the student, to the extent possible, to solve problems independently.

Referring to Figure 9, the first hints classify the problem and direct the student to pertinent readings. The second hint points to a pertinent example, mentioning the similarity between the problem and the example. The subsequent hints are more specific to the problem. There are usually a dozen or fewer hints that address a majority of the questions students ask about a particular problem. The hints can be refined by tracking student comments and questions during the early stages of a problem's use. This ensures that the hints provided in a particular problem are as helpful as possible.

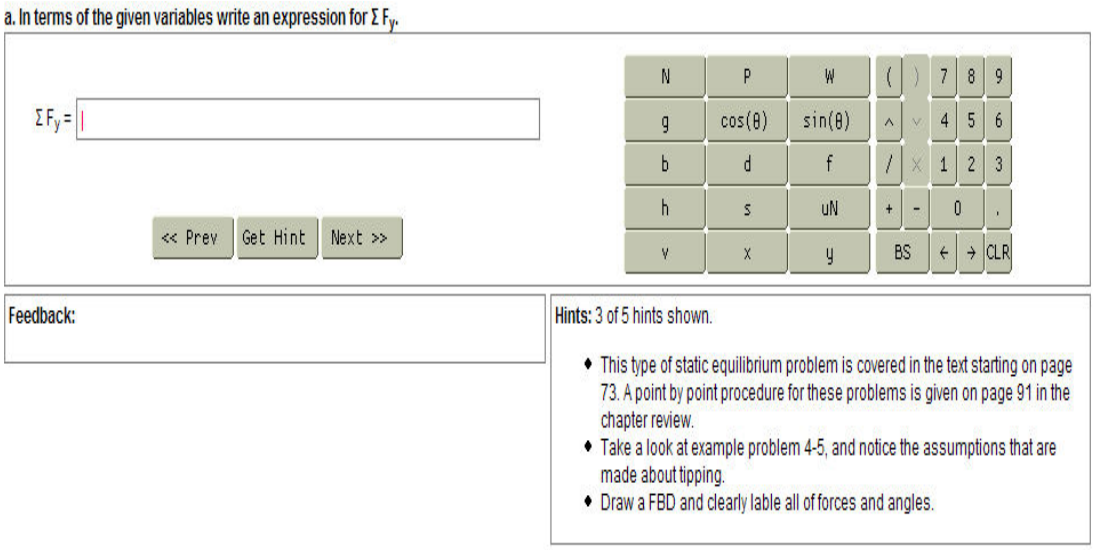


Figure 9: Guidance with hints

4.5 Detailed Feedback

Providing the student with detailed feedback is technically a more difficult problem than providing effective hints. The following shows that the problem of detailed feedback has been solved in Expert TA. Instructors frequently find it difficult to locate mistakes, particularly small errors like sign errors, because the expression can be expressed in a variety of ways. The expressions are written in open-form and there can be a page or more of work to look through. This was also a technical problem in online systems. Figure 10 shows a close-up of the interface portion of an example problem in Expert TA. The correct expression for the resultant forces in the y direction is $\Sigma F_y = N - W - P \sin\theta$. The form of the expression that was entered in Fig. 10 is absurdly complicated to make the point that with open-form expressions the student can enter an expression in any form. Expert TA

recognizes the expression entered regardless of the form. The recognition of the expression is one of the features of the expression analyzer (See Appendix A). As shown, Expert TA deals with the form of an entered expression, identifies each mistake made and gives feedback. Notice that the feedback for each term is reported to the student in its "normalized" form. The expression is normalized in the background as a preliminary step that is performed before determining detailed feedback. For example, notice in Fig. 10 that the system recognizes that (((*N*))) (-1) is the same as just – *N* and that cos(*θ*)*P* W^{g-g} reduces to *P* cos(*θ*).

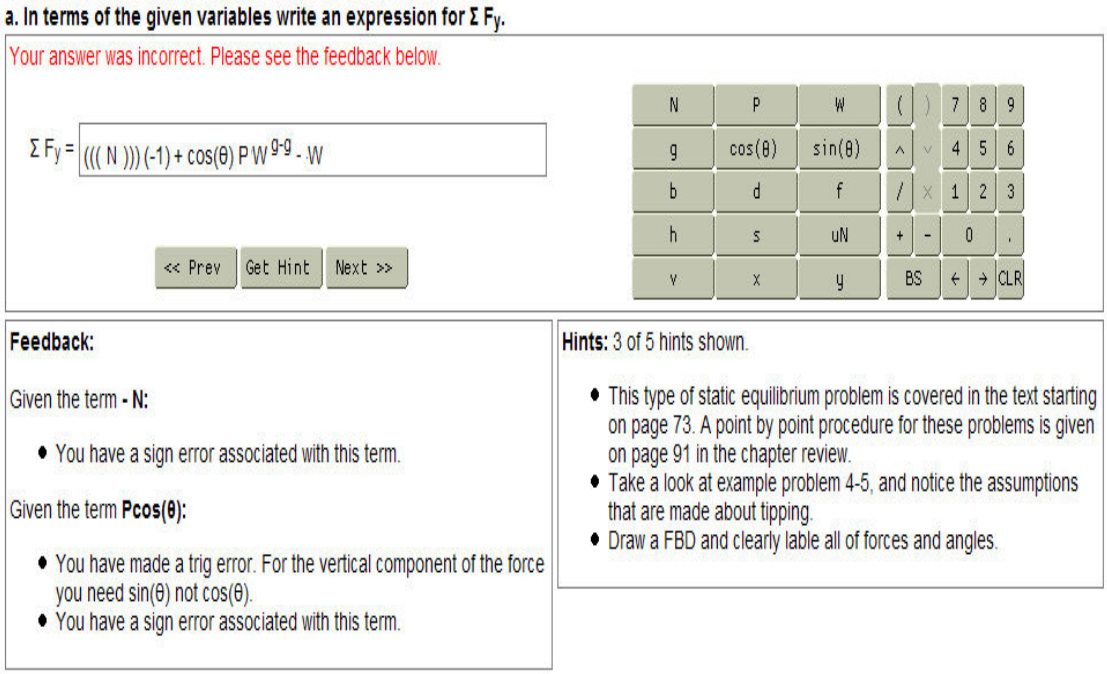


Figure 10: Detailed feedback

The detailed feedback addresses all of the mistakes made in the expression entered. The incorrect terms and the extra terms found in the normalized form of the expression are reported. Table 1 contains further examples of expressions entered versus the correct expression and the corresponding feedback.

Table 1: Treatment of expressions

Correct Expression	Student's Expression	Feedback
□ $F = a\cos(\theta) - b\sin(\phi)$	□ $F = -b\sin(\theta)(-d/d) + a\cos(\phi)$	**For the term: $b\sin(\theta)$** You have a sign error. You have a trig error; you should have $\cos(\theta)$ not $\sin(\theta)$.
□ $F = P_1\cos(\theta) + P_2 - \mu_s N$	□ $F = P_1\cos(\theta) - \mu_s N$	**For the term: P_2** You are missing this term. Please check your free body diagram.
□ $F = a\cos(\theta) - b\sin(\phi)$	□ $F = P - a\cos(\theta) + F - b\sin(\phi)$	**For the term: a $\cos(\theta)$** You have a sign error. **For the terms: P , F** Both P and F are extra terms. These terms are not part of the correct expression.
$F = 1.12$ lb	$F = 1.120$ lb	**Significant Figures** You have the wrong number of significant figures. Express the answer using three significant figures.

It should be noted that the detailed feedback illustrated in Table 1 is automated. The author of a problem, during problem creation, is not required to input anything other than the correct expression (See Appendix). In general, an expression is deconstructed into an exploded super-set of terms with a topology.

4.6 Student Assessment

The student does not have access to hints and feedback is not given for mathematical mistakes when Expert TA operates in homework mode (when default settings are used) (See Fig. 8). Like when submitting homework manually, the student completes each problem in an assignment and submits his or her work after completing the assignment. The student, immediately after the submission, receives a numerical grade for each problem and the entire assignment. A detailed grade report is returned after the due date of the assignment has expired because it contains the full solution to each problem.

Figure 11 shows the grade report for the problem shown in Fig. 8. A Student had entered the expressions in any form and the expression analyzer reduces them into simplified forms. The simplified forms are compared with the correct expressions on a term-by-term basis. For clarity, the original expression that the student had entered, the simplified expression, and the correct expression are displayed in each part of the problem.

Problem 2-41

Part A:

Correct Answer: Wb cos(θ) - Wc cos(φ)

Student Answer (Original): ((Wc)) cos(φ) + Wb sin(θ)

Student Answer (Simplified): Wc cos(φ) + Wb sin(θ)

Correct Answer	Student's Answer	Deductions	Feedback
Wbcos(θ)	Wbsin(θ)	- 16.65	Examine your geometry. For the horizontal component of force you need cos(θ) not sin(θ).
- Wccos(φ)	Wccos(φ)	See below.	You have a sign error associated with this term.
Sign Error Penalties	-15		
Extra Term Penalties	0		
Total Credit			**68.35 out of 100**

Part B:

Correct Answer: Wb sin(θ) + Wc sin(φ) - W

Student Answer (Original): g^{x-x} W + Wb sin(θ) + ((Wc)) (sin(φ) + 1) - Rc - Wc

Student Answer (Simplified): W + Wb sin(θ) + Wc sin(φ) - Rc

Correct Answer	Student's Answer	Deductions	Feedback
Wbsin(θ)	Wbsin(θ)		
Wcsin(φ)	Wcsin(φ)		
- W	W	See below.	You have a sign error associated with this term.
	- Rc	See below.	The student earned no credit for this extra term.
Sign Error Penalties	-15		
Extra Term Penalties	-15		
Total Credit			**70 out of 100**

Part C:

Correct Answer: 65.86 Lb

Student Answer: 66.15 Lb

Correct Answer	Student's Answer	Deductions	Feedback
65.86 Lb	66.15 Lb		
Total Credit			**100 out of 100**

Part D:

Correct Answer: 310.15 N

Student Answer: 310.5 N

Correct Answer	Student's Answer	Deductions	Feedback
310.15 N	310.5 N	- 25	You have the wrong number of Sig Figs.
Total Credit			**75 of 100**

Figure 11: Grade report for a typical problem

The grade report gives a term-by-term breakdown of an expression below each part of a problem. The report shows the specific deductions and detailed feedback for each mistake. The report emulates the hand-graded solution shown in Fig. 7 except that it's in tabular form.

Notice that some of the deductions are displayed at the bottom of the table. These are the deductions that are not associated with a single term. For example, sign errors are not necessarily associated with a single term; they can be associated with summing forces in the wrong direction, which effects all of the terms in an expression. Parts C and *D* of the grading report are parts of the solution that request a short response, i.e., numerical values and units. The answers are treated as correct when they are accurate to three decimal places.

Like an instructor, online grading follows a grading rubric. Deductions are made for each mistake depending on a significance that is attributed to the mistake. It is at this stage of the grading process in which it is common to find differences between teaching styles among instructors. Some instructors deduct more points than others for certain types of mistakes. Figure 12 shows where an instructor sets his individual preferences. As shown, the mistakes that are found in an expression are divided into types and the instructor sets his preferences for each type.

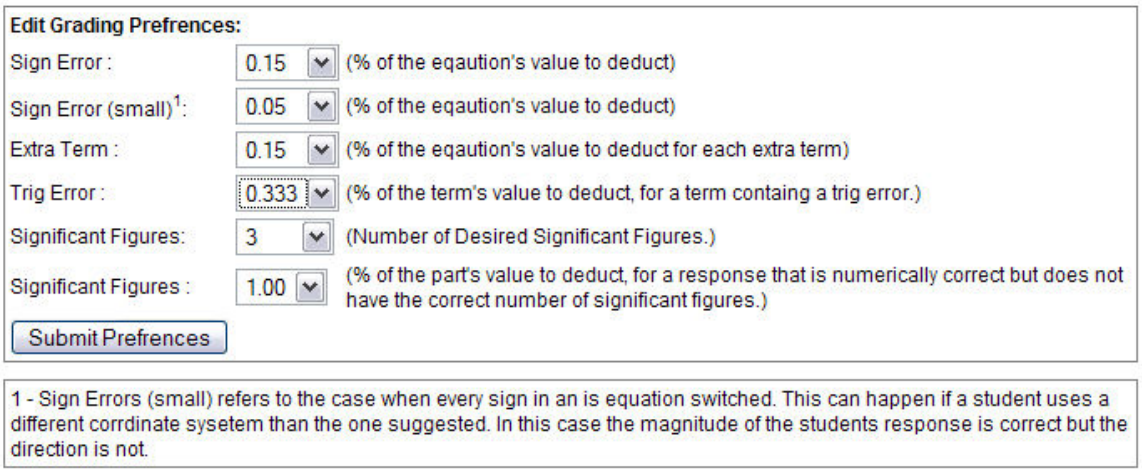


Figure 12: Setting an instructor's preferences

Online grading emulates manual grading. In terms of desirable grading features, the online system locates the specific terms in an expression that are in error, provides feedback in response to errors, and follows a consistent grading rubric. In engineering, an instructor's

responsibilities divide into helping students and grading. The time and resources saved by eliminating manual grading can be substantial.

4.7 Class Assessment

The expression analyzer used in Expert TA breaks down each expression a student enters into terms and relationships. It can thought of as providing an elaborative data base used for finding mistakes, providing detailed feedback, and grading. But it can be used for other instructional purposes that would be impractical without a computer. Indeed, once detailed information of this nature is available online, detailed instructional analysis can be performed, not just for individual students, but for the class as a whole. A detailed discussion of the different tools that can be developed using the data base is beyond the scope of this dissertation, but the following mentions several tools that are made possible.

(*a*) Online Assessment for Accreditation: The types of errors in each problem can be categorized to break down class proficiency by the objective measures used in engineering accreditation. The high detail of the data base and the ability to evaluate every student and not just a sample makes online assessment potentially very powerful. The collection of data has always been a serious challenge in accreditation.

(*b*) Student Habits: There is a strong correlation between time management skills and performance. Individual students with poor time management skills and poor performance can be identified and attention given to them to improve their study habits.

(*c*) Positive Reinforcement: A student can be allowed to turn in the same homework set more than once to provide him the opportunity to correct his or her mistakes (if this takes place before the homework deadline). This encourages the student to more thoroughly correct his own work, and it promotes good time management practices.

(*d*) Class Inspections: An instructor can inspect the performance of his class when an online system is used. The problems that the students are having trouble with can be flagged, and the common errors identified. The instructor can then make in-situ instructional modifications in response to common errors that are identified.

Chapter 5. Pedagogic Summary

This dissertation began with a discussion leading to the hypothesis that the treatment of expressions is critical to the adoption of online systems in engineering classes. This hypothesis leads to several critical questions. Is it even feasible to ask students to enter expressions into online systems from the viewpoint of the human factors that are being imposed on the student? Secondly, once expressions are entered, and assuming that software has been developed to breakdown and analyze expressions, what kind of benefits can be gained? How can online systems help tutoring and assessment? What kind of detailed feedback can be offered beyond simple things like the answer is right or wrong? Can grading of student work be done online and what about class assessment?

The dissertation began by explaining why expression entry has become a critical problem in online systems for engineering classes. Online systems can handle problem statements of the form multiple-choice and true-false, but when student input is in the form of expressions human factors had imposed prohibitively complex requirements. The problem of student input of expressions was cast as an optimization problem in which the time to enter an expression and the special knowledge required on the part of a student are minimized. Through a critical review of the state-of-the-art of online systems, the challenges faced with data entry were identified. The review identified ergonomic problems with general syntax help, problem-customized syntax help, and the customized expression editor on the basis of navigation time and special knowledge on the part of the student. Figure 13 summarizes the expression entry approaches.

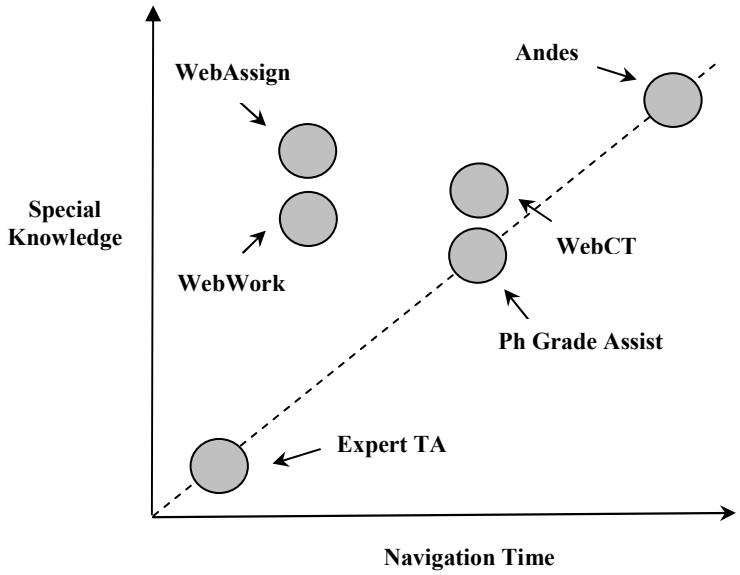


Figure 13: Qualitative Comparison of Online Assessment Systems Based on Navigation Time and Special Knowledge for Expression Input

The major ergonomic problems found in the review were remedied in Expert TA through the use of a problem-customized, integrated expression editor (palate). The problem customized buttons eliminated the need for special knowledge and hence the need for help menus to explain syntax. The back and forth motion between a mouse, keyboard, and an expression editor were eliminated because the motion was reduced to clicks on a palate. Syntax errors were eliminated through the development of a method called error-proof expression entry, in which operation buttons are dynamically disabled during expression entry. Finally, a keyboard auto-complete feature was introduced for students who prefer using a keyboard during expression entry. In summary, the system employed in Expert TA eliminates the causes of excessive navigation time and eliminates the need for special knowledge on the part of the student.

To understand how online tutorial systems can help the student and the instructor and potentially offer advances that go beyond traditional practices, this dissertation reviewed the traditional relationship between the student and the instructor. The traditional relationship was characterized as one in which an instructor provides a) guidance through hints and b) feedback in response to mistakes. This was compared with a review of the current state-of-the-art. It was found that the most basic online systems offer problems divided into steps, and

questions of the form multiple choice, true or false, and multiple select. Very few dynamic hints or feedback are offered except whether an answer is correct or not. No ability to find a student mistakes existed. Through the development of an expression analyzer that operated in the background, Expert TA provides detailed feedback in response to specific mathematical mistakes. This emulates traditional practices. A significant portion of the dead time that students would otherwise spend waiting for help when they are stuck is therefore eliminated and the difficulty and time spent by instructors locating the origin of a student's mistake is also eliminated.

Best practices in grading was described, in order of decreasing importance, a) finding where each mistake is made, b) providing specific feedback about each error, and c) grading consistently. Emulating best practices, Expert TA was shown to be able to find mathematical mistakes that a student makes and follow a grading rubric that is based on the specific mathematical mistakes made. The main issues that have been addressed are summarized in Table 2.

Table 2: Pedagogic Summary Table

Issue	Problem Statement	Solution to the Problem
Special Syntax Knowledge	The need for the student to have special syntax knowledge compromises the instructional effectiveness of online systems.	The need for specialized syntax knowledge is eliminated using a problem-customized, integrated expression editor, called a palate. Using the palate, all of the symbols needed in a problem are in full view.
Careless Data Entry Mistakes	Careless data entry mistakes, like misplaced parentheses, are common. Debugging is generally required to find and eliminate the mistakes. This is time consuming and extraneous to the problem-solving process.	The button's in the palate are dynamically enabled and disabled to prevent invalid expressions from being entered. This eliminates data entry mistakes.
Navigation Time and Frustration	The complex back-and-forth motion between an expression editor's modules and the keyboard is both time consuming and frustrating.	The palate has no modules. Every function and symbol needed for a problem is contained in the palate. The need for navigation and back-and-forth motion are eliminated.
Learning Curve	A learning curve is required to navigate between modules.	No learning curve is required using the palate.
Interference with Instruction	Online systems contain ergonomic features that interfere with instruction, namely, special knowledge, careless data entry mistakes, and excessive navigation time.	All of the ergonomic features that interfere with instruction are eliminated through the use of the palate.
The Form of Questions	Online systems ask questions in the form of short response, multiple choice, and true-force. This is different than the traditional engineering practice of grading expressions.	Following traditional engineering practices, an expression analyzer was developed that grades expressions.
Finding Errors	Instructors, when grading assignments, locate the errors. This is important because the student can then see what mistakes were made and how to correct them in the future.	The expression analyzer locates the specific terms in expressions that are incorrect.

Table 2 Continued

Issue	Problem Statement	Solution to the Problem
Providing Detailed Feedback	Instructors, when grading assignments, provide helpful feedback about mistakes found. This also helps the student understand the mistake to prevent it from recurring.	Using the expression analyzer, detailed feedback is given that describes the mistake, e.g., sign error, used sin instead of cosine.
Handling Open-Form Expressions	A single expression can be mathematically expressed in different forms. It is hard to know and it takes time to check whether the expression entered is correct.	The expression analyzer reduces every expression that is entered into a unique simplified form. Thus, the expression recognizes any correct form that has been entered.
Grading Consistency	Instructors, to grade consistently, divide solutions into steps and use grading rubrics that deduct points for different types of mistakes. Still, grading consistently is difficult.	Online grading also divides solutions into steps and uses a grading rubric. The grading rubric, since it's an algorithm, is consistent. The instructors specify the levels of deduction for the different types of mistakes, depending on their preferences.
Generating Problems	It is important that the time and effort required to generate a problem online be minimal. Otherwise, the practicality of the system is called into question.	GUIs (Graphical User Interface) eliminate the writing of code. Detailed feedback and grading are automated using the expression analyzer, and hence do not add complexity to problem generation.
Effective Hints	To be effective at offering hints, experience is needed. Generally a small number of hints is sufficient and the instructor doesn't know which one's to provide in advance	Effective hints are developed in an iterative process that responds to questions that students ask about a particular problem. The hints become static content once the statistics associated with the most frequently asked questions are stabilized.
Grading Time	To grade effectively, the grader needs to locate each of the students' mistakes and provide detailed feedback that explains them. This is time consuming; in some cases making up half of the tutor's time.	Automated grading and detailed feedback eliminates the time otherwise spent on grading. The time saved can be spent on offering more office hours, giving problem review sessions, refining lecture notes, developing class experiments, etc.
Student Down Time	Students, when completing assignments, get stuck and need help. The down time associated with waiting until help is available is considerable. This is a serious impediment to student learning.	Detailed feedback that locates errors is used by students when they're otherwise stuck. Online access enables the student to reduce his down time considerably.
Changing Standard Practices in Engineering Pedagogy	The access that online systems offer is wonderful but not helpful when it doesn't follow standard practices. In engineering, problems are mathematical and instructors assess the process the student follows, not just the final answer.	The expression analyzer was developed specifically to find mistakes within expressions to enable a student's work to be assessed, not just the final answer. Standard practices in engineering pedagogy are emulated.
Instructor Time of Finding Mistakes	Whether grading a student's work or sitting down with a student to find out at what step a student is faltering, finding a student's mistake is a time consuming task.	The detailed feedback discloses where the mistake is made and why. The time required of a student or an instructor to find the reason behind the mistake once this information is provided is comparatively short.

Chapter 6. Mathematics and Technology

The primary goal of this research has been to develop an educational tool that meets the needs of the engineering community. As such the principle elements discussed thus far have been from the standpoint of pedagogy and application in an educational setting. However, it is important to note that the significant advancements made here are on the level of mathematics and technology. Long before the advent of the computer, and to this day, educators have utilized the approach of analyzing a student's work by hand, identifying mistakes, and grading with partial credit. Developing alternative computer base techniques would be tantamount as to evacipate traditional pedagogy. Computerized partial credit grading of expressions did not exist prior to this research, due to mathematical and technological limitations in software, and not due to pedagogical shortcomings. The following two sections address the mathematical and technological aspects of this research that are fundamental to the function of this educational tool.

6.1 Mathematics

The following describes the expression analyzer used in the background of Expert TA. The expression analyzer uses custom-built algorithms that simplify an expression, and deconstruct it into reduced terms and relationships, producing a unique normalized expression. The terms are compared with an equivalent form of the correct expression during which specific errors associated with terms are identified, detailed feedback is given, and consistent awards of partial credit are calculated. This section describes the general process followed by the expression analyzer.

6.1.1 Expression Normalization and Symbolic Analysis

Imagine the grading situation where the correct answer is $y = x - 1$ and the student has input $y = (3x^2 - 5x + 2)/(3x - 2)$. Perhaps there is room for discussion as to whether the student should be penalized for not simplifying, but what is clear is that the student does have the correct answer.

The goal is to maintain resolution on a symbolic level so that information is not lost as is the case with expression evaluators. An expression such as this can easily be recognized as

correct by using an expression evaluator, but at the cost of loosing information. Essentially, an expression evaluator assigns a numerical value to every variable or symbol in an expression and evaluates the expression to a numerical result. The same is done for the correct expression and the two numbers are compared, with some accepted tolerance, to determine if the expressions are equivalent. For grading purposes expression evaluators are useful only to determine if an expression is right or wrong.

Imagine that the student had input $y = (3x^2 + x - 2)/(3x - 2)$ which is equivalent to $y = x + 1$. Grading this expression with an expression evaluator will result only in the information that the expression is not correct. All information contained within the expression is lost during numerical compression. As a result, numerical analysis is not conducive to grading expressions in detail. Clearly symbolic analysis is required.

The first step in detailed grading is to get the input expression into some normalized form prior to making a comparison. If the given expression is first symbolically simplified from $y = (3x^2 + x - 2)/(3x - 2)$ to $y = x + 1$ a proper comparison can be made.

One of the obstacles that online systems faced, and which is overcome by Expert TA's expression analyzer, is the methodical treatment of open-form expressions. Expert TA uses algorithms that reduce any expression into a unique form, called the normalized expression. The normalized expression is later compared with a correct expression provided by the author of a problem. (The expression analyzer also reduces the author's expression to a normalized form so actually two normalized forms are being compared.)

Mathematical packages like Maple simplify expressions with different objectives than Expert TA's expression analyzer. For example, the Maple function *simple* "produces different simplifications and then chooses the shortest [30]". Maple functions can not be used to reduce mathematical expressions to a form that satisfies the requirements of locating errors in an expression and assigning partial credit. Therefore, it became necessary to develop a custom-built expression analyzer. Theoretically, it was found how to produce a unique normalized expression. The normalized expression has a unique combinatorial sub-space

structure defined in terms of mathematical operators and separators in a problem step. For example, the operators and separators in Fig. 8 are +, -, /, x, (,), and ^.

A fundamental premise is that there are a finite number of unique expression structures that can be constructed with a finite number of mathematical operators and separators. In combinatorial mathematics the topic of Catalan numbers is directly related to this concept. The following expression is given for calculating the n^{th} Catalan number.

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{for } n \geq 0. \tag{1}$$

"The book *Enumerative Combinatorics: Volume 2* by combinatorialist Richard P. Stanley contains a set of exercises which describe 66 different interpretations of the Catalan numbers"[31]. The following are two of the examples posed in the book with n = 3 [32].

- C_n counts the number of expressions containing n pairs of parentheses which are correctly matched:

  (((()))) ()(()) ()()() (())() (()())

- C_n is the number of different ways $n + 1$ factors can be completely parenthesized:

  $((ab)c)d \quad (a(bc))d \quad (ab)(cd) \quad a((bc)d) \quad a(b(cd))$

This schema here is expanded to encompass expressions that are formed from variables and all of the available operators (not just parentheses). When this is done it is clear that only a finite number of unique expression structures can be formed. Groups can then be formed of structures that are similar, and that will behave predictably under the same transformations. With these distilled structural subsets the simplification of an expression is then manageable. If one knows a list of all possible structures in which an expression can be written transformations for each structure, or group of structures, to manipulate that expression into a desired form can be formulated.

When an expression is being normalized it is not allowed to get stuck in a finite sub-space, lest resulting in a non-unique form. Expert TA handles this by applying a series of transformations designed to target each sub-space. In short, one of the subspaces is chosen as the preferred sub-space. When an expression tries to simplify to any other sub-space, transformations are applied that map the expression to the desired subspace. Figure 14 shows a few selected normalization steps of the expression in Fig. 8.

Normalization

Functions	Equation
Equation before Pre-Parse:	$-Pf+(((((-1)))))W+Nx^{y-y}+P(f-\cos(\theta))$
Regular Expression 1:	$-Pf+(((((-1)))))W+Nx\,\hat{}\,y-yv+P(f-\cos\{\theta)$
Regular Expression 2:	$-Pf+(((((-1)))))W+Nx\,\hat{}\,y-yv+P(f-\cos\{\theta\})$
Class A Parenthese:	$-Pf+(((((-1)))))W+Nx\,\hat{}\,y-yv+P(f-\cos\{\theta\})$
Number Format:	$-Pf+(((((-1)))))W+Nx\,\hat{}\,y-yv+P(f-\cos\{\theta\})$
Exponent Preformat:	$-Pf+(((((-1)))))W+Nx\,\hat{}\,(y-y)v+P(f-\cos\{\theta\})$
Class B Parenthese:	$-Pf+(-1)W+Nx\,\hat{}\,(y-y)v+P(f-\cos\{\theta\})$
Exponent:	$-Pf+(-1)W+N1+P(f-\cos\{\theta\})$
Expand:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Division:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Strip Brackets:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Expand B:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Secondary Expand:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Paren Hold:	$-Pf+W(-1)+N1+P(f-\cos\{\theta\})$
Distribution:	$-Pf-W1+N1+Pf-P\cos\{\theta\}$
Numeric:	$-W+N-P\cos\{\theta\}$
Secondary Strip Brackets:	$-W+N-P\cos\{\theta\}$
Equation after pre-parsing:	$-W+N-P\cos(\theta)$

Figure 14: Example of Normalization

6.1.2 Expression Deconstruction

The process of normalization is only a preliminary step in the analysis of an expression. After being normalized, the expression is deconstructed into a set and relationships. A simplified illustration of the deconstruction process is shown in Fig. 15. As shown, a normalized

expression is required at the beginning of the deconstruction. A structural transformation performs a decomposition that deconstructs an expression into logical groupings, called terms. For example, the expression $y = a\cos(x) + b\sin(y)$ deconstructs into the two terms $a\cos(x)$ and $b\sin(y)$. The deconstruction process is by nature recursive but finite. For example, the expression $y = a + (b\cos(x) + c)^{\sin(y)}$ is first deconstructed into the term a, the base $b\cos(x) + c$, and the exponent $\sin(y)$. The base $b\cos(x) + c$ is a complicated term meaning that it can be deconstructed itself into more than one term. The base $b\cos(x) + c$ deconstructs into the two terms $b\cos(x)$ and c. The deconstruction ends when all of the terms are simple. Ultimately, the normalized expression deconstructs into a set with a specific structure and attributes associated with terms. The comparisons are made between simple terms while considering the set's structure and attributes to determine the feedback that the student is offered and the partial credit awarded (in the homework mode).

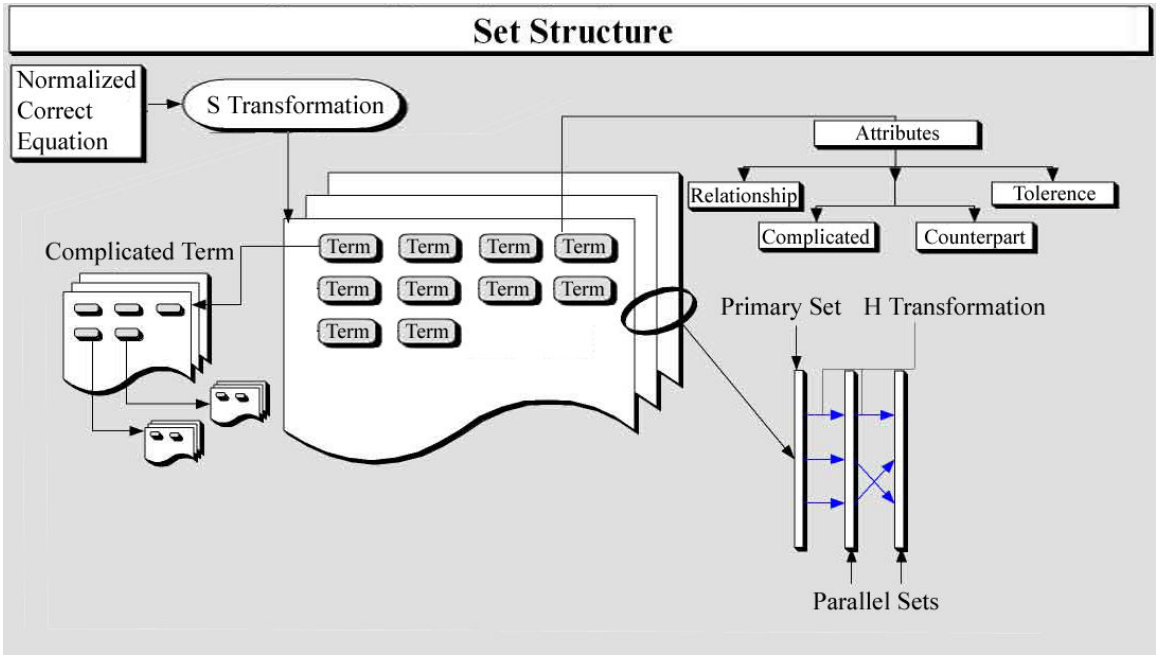


Figure 15: Deconstruction of a Normalized Expression

6.2 Technology

6.2.1 PHP

The web-application was implemented predominantly in PHP. PHP is a web friendly language that offers considerable power in the background[33]. PHP is an interpreted language with syntax and functionality similar to C++. In recent article in the Journal of Online Mathematics and its applications the use of PHP in higher education mathematics classes is discussed [34]. The overall conclusion reached is that PHP is versatile, powerful, and a viable choice of languages for use in higher level math classes.

6.2.2 MySQL

Databasing is almost always needed for a web service. General uses for the database are storing user ids, passwords, and preferences. A much larger and more complicated databasing structure is required for Expert TA due to the nature of its purpose. Large amounts of data need to be stored pertaining to classes, assignments, problems, start and stop times of assignments, etc… In addition all of the students grades must be stored, as well as all of the privileges and relationships that specify who has access to what information. MySQL was used for all of the databasing in Expert TA. MySQL is an industry standard and determined to be the appropriate tool for the job [35].

6.3 Selected Functions for Normalization

The expression analyzer is comprised of two primary sections: algorithms for expression normalization and for expression deconstruction. By far the largest portion of code for the expression analyzer is that associated with expression normalization, which has acquired the label “pre-parse”. Pre-parse consists roughly of 43 functions each which provide a number of transformations that target specific combinatorial structures of an expression. At present pre-parse is roughly 7800 lines of code in length which translates to about 260 pages of text, and will therefore not be presented in full. Rather, a general discussion of the mathematics associated with selected functions is given here and the source code for those selected functions is provided in Appendix A.

It was irrefutably demonstrated earlier that symbolic simplifying expressions is necessary to grade those expressions in detail. Numerical analysis is only conducive to the right or wrong

grading of expressions. The primary source of complexity when performing symbolic manipulation stems from the large number of possible forms in which an expression can be entered.

The two functions presented target simple factorization and the factorization of polynomials. These two functions were chosen because they are relatively short yet demonstrate how involved the logic and mathematics can become in symbolic analysis, even for simple tasks.

6.3.1 Simple Factorization

The process of simple factorization is an example of a task that is relatively easy for a human to do but that requires a reasonable amount of logic to accomplish algorithmically. To put this problem in perspective consider the following expression.

$$F(x) = a\ b\ x\ c + x\ d\ e - f\ g\ x\ h + x \qquad (2)$$

By a person, the factorization process is carried out by simply scanning the terms for common elements. In this case all of the terms contain and "*x*". Having identified the common element the expression can be rewritten as $F(x) = x\ (a\ b\ c + d\ e - f\ g\ h + 1)$. While this process can be done quite readily by a person it is logically "dense" in algorithmic form.

The PHP function that handles simple factoring is presented in Appendix A. As can be seen the process initially consists of four nested for loops. The first step is to break down the expression into an array that contains all of its terms. Each term must then be exploded into an array containing its parts. For example the term "*a b x c*" has the parts "*a*", "*b*", "*x*", and "*c*".

For each part of each term you must loop through all of the remaining term's exploding each term into its respective parts and searching for a match. It must be kept in mind that an element is only a factor if it is found in all of the remaining terms. As this process is done a mapping of the relative and global locations of matching variables must be generated in the event that the variable is indeed a factor. The mapping contains the location of each of the unwanted parts for each term in the expression. This information is necessary since the

expression will have to be rebuilt without each factor. As can be seen from the source code, the process is not trivial.

6.3.2 Polynomial factorization

The fundamental theorem of algebra states that every polynomial of degree n is factorable into the form $p(z) = (z - z_1)(z - z_2) \ldots (z - z_n)$. This form of polynomial factorization does not ensure that that z_1 through z_n will be integers. As discussed earlier it is necessary to be able to simplify the expression $(3x^2 - 5x + 2)/(3x - 2)$ to $(x - 1)$. Standard polynomial factorization as described above is not formulated to provide such simplification. To accomplish this form of simplification a method for 'cleanly' factoring polynomials is required.

Suppose that we have the polynomial $A\,x^2 + B\,x + C$ where A, B, and C are integers and are know. If the polynomial is factorable it will have the form $(ax + b)(cx + d)$. This yields the following relationship

$$A\,x^2 + B\,x + C = (ax + b)(cx + d) = ac\,x^2 + (ad + bc)\,x + bd. \tag{3}$$

Equating the coefficients yields

$$A = ac \tag{4}$$

$$B = (ad + bc) \tag{5}$$

$$C = bd \tag{6}$$

The question at hand is "Is there a combination of integer values for a, b, c, and d such that the above expressions are satisfied?" The solution to this question is presented in Appendix A. The general format of the code is an "if" statement nested in a foreach loop, nested in a foreach loop. The first step is to determine the prime factorization of the coefficient A and all possible combinations of those factors. The outside foreach loop goes through each possible integer "*a*" and "*c*" that multiply to equal "*A*". Of course all possible integer values of "*a*" and "*c*" that multiply to equal "*A*" are simply the permutations of the prime factorization of "A". For each of these, the inner loop goes through each integer "*b*" and "*d*" that multiply to equal "*C*" (Again "*b*" and "*d*" are combinations of the prime factors of "*C*"

). The innermost "if" checks the final condition. By definition $ac = A$ and $bd = C$. If $(ad + bc) = B$ then the polynomial is factorable. If there is not combination of integers such that the above condition is satisfied then the polynomial is not factorable.

The given code consists of four functions. The polynomial factorization function takes the coefficients of a second degree polynomial. The function "primeFactors($number)" is used to determine the prime factorization of each of the integer coefficients. The function "getFactorsPerms($factors)" takes an array representing the prime factors of a number and returns an array of all the possible combinations of those factors. The function "resolvePerms" takes the permutations and generates the numerical factor obtained by multiplying the integers together.

6.4 Mathematics and Technology Summary

The technological tools must be accepted industry standards and the right tool for the job, but are just a means to an end. The selected tools were PHP and MySQL which were determined to be the right tools for the job at hand.

The process of grading an expression in detail is built upon a highly organized mathematical structure. The first step in grading a free-form expression is to simplify that expression into a normalized form that is appropriate for comparison. The earlier mathematical discussion of normalization and the two processes presented above for simple factoring and factoring of polynomials, give a feel for what is entailed in symbolic normalization. Once the expression is simplified it is deconstructed into a set formulation with a topology, and the grading process is, in effect, taking the intersection of the correct set and the student's set.

As discussed earlier the mathematics and technology were not the driving factors in this research. The goal of this research was to develop an educational system that could emulate the methods of an instructor for both tutorial practices as well as partial credit grading. For this the need to analyze expressions in detail and to be able to identify individual mistakes became a clear requirement. This warranted the construction of the mathematical structure, which now serves the purpose quite well.

References

[1] J. Morton, L. Silverberg. "Building Online Systems for Engineering Classes: Student Input of Expressions". *Journal of Engineering Education*. Submitted February 2006.

[2] T. Waits, L. Lewis, B. Green Distance Education at Degree-Granting Postsecondary Institutions: 2001-2002. National Center for Educational Statistics and the U.S. Department of Education. pp 11-15,

[3] Lilian Cao and Golgen Bengu. "Web-Based Agents for Reengineering Engineering Education" *Journal of Educational Computing Research.* 2001 Vol 23. Num 4. pp. 421-430.

[4] Richard Felder, and Rebecca Brent. "Designing and Teaching Courses to Satisfy the ABET Engineering Criteria" *Journal of Engineering Education,* 2003 *Vol 92. Num* 1. pp. 7-25.

[5] John Dutton, Marilyn Dutton, Jo Perry. "How Do Online Students Differ From Lecture Students?" *Journal of Asynchronous Learning Networks*. July 2002 Vol 6. Num 1.

[6] K.C. Chu. "The development of a Web-Based Teaching System for Engineering Education" E*ngineering Science and Education Journal.* 1999 Vol 8. Num 3. pp. 115-118.

[7] Kelvin Cheng, Beth Thacker, and Richard Cardenas. "Using an online homework system enhances students' learning of physics concepts in an introductory physics course" *American Journal of Physics* -- November 2004 -- Volume 72, Issue 11, pp. 1447-1453.

[8]VanLehn, Lynch, et al. "The Andes Physics Tutoring System: Lessons Learned". *International Journal of Artificial Intelligence in Education*, 15(3), 2005.

[9] Michau, Gentil, and Barrault. "Expected benefits of web-based learning for engineering education: examples in control engineering" *European Journal of Engineering Education.* June 2001 Vol 26. Num 2. pp. 151-168.

[10] Gramoll, K. C. "Teaching Statics Online with Only Electronic Media on Laptop Computer", in the Proceedings of the ASEE 1999 Annual Meeting, Charlotte, NC, CD-ROM Session 1668.

[11] John Bourne, Dale Harris, and Frank Mayadas. "Online Engineering Education: Learning Anywhere, Anytime". *Journal of Engineering Education.* January 2005 Vol. 94, No. 1, , pp. 131–146.

[12] Charles Harrington, and Saxon Reasons. "Online Student Evaluation of Teaching for Distance Education: A Perfect Match?" *The Journal of Educators Online*. January 2005 Vol 2. Num 1.

[13] Catalano, G.D., Catalano, K.C. "Transformation From Teacher-Centered to Student-Centered Engineering Education", *Journal of Engineering Education*, vol.88, no.1, 1988, pp.59-64

[14] Shandy Hauk and Angelo Segalla. "Student perceptions of the web-based homework program WeBWorK in moderate enrollment college algebra classes" J*ournal of Computers in Mathematics and Science* submitted march 2004.

[15] http://www.webassign.net

[16] VAnLehn, Schulze, Shapiro, et al. "The Andes Physics Tutoring System: Lessons Learned" International Journal of Artificial Intelligence in Education, vol.15, no.3, 2005.

[17] http://www.webct.com/expression

[18] http://www.prenhall.com

[19] Assessing Higher Mathematical Skills Using Computer Algebra Marking Through AIM Christopher James Sangwin (need to find this formally)

[20] http://www.andes.pitt.edu/

[21] http://www.webmentor.com

[22] http://www.brainbench.com

[23] http://www.virtual-u.org

[24] http://www.masteringphysics.comm

[25] J. L. Merriam & L. G. Kraig (1997) Engineering Mechanics: Statics Fourth Edition. John Wiley and Sons.

[26] J. L. Merriam & L. G. Kraig (1997) Engineering Mechanics: Dynamics Fourth Edition. John Wiley and Sons.

[27] Roy R. Craig. (2000) Mechanics of Materials Second Edition. John Wiley and Sons.

[28] Bruce R. Munson, Donald F. Young, & Theodore H.Okiishi. (1998) Fundamentals of Fluid Mechanics Third Edition. John Wiley and Sons.

[29] Shapiro, J. A. (in press). Algebra subsystem for an intelligent tutoring system. *International Journal of Artificial Intelligence in Education*.

[30] Sigmon K. 1994. Matlab Primer Fourth Edition. Boca Raton, United States: CRC Press

[31] http://www.wikipedia.com

[32] Stanley, R. P., (1999) *Enumerative Combinatorics: Volume 2.* Cambridge

[33] http://www.php.net

[34] Bennett, A. (2002), Using PHP for Interactive Web Pages, *Journal of Online Mathematics and its Applications*, **2**

[35] http://www.mysql.com

Appendix

Appendix A: Selected Normalization Functions

A.1 Simple Factorization

```
<?php
require_once("Splitpm2.php");
require_once("Spacing.php");
require_once("Sortterms.php");
require_once("Numeric.php");

function factor($oldString)
{
        $tallySign = 0;
        for ($i=0; $i<strlen($oldString); $i++)
        {
            if($oldString[$i] == "+" || $oldString[$i] == "-")
            {
                $tallySign++;
            }
        }

        if ($tallySign != 0)
        {
            $oldString = splitpm2($oldString);

            $unWantedI = array();
            $unWantedJ = array();
            $newFactor = array();

            for ($i=0; $i<count($oldString); $i++)
            {
                if (trim($oldString[$i]) != "+" &&
                trim($oldString[$i]) != "-")
                {
                    $placeI = array();
                    $placeI[] = $i;
                    $tempPart = trim($oldString[$i]);
                    $tempParts = explode(" ",$tempPart);

                    for ($j=0; $j<count($tempParts); $j++)
                    {
                        $unWantedITemp = array();
                        $unWantedJTemp = array();

                        $tallyMatch = 0;

                        $placeJ = array();
                        $placeJ[] = $j;

```

```
                    if(
                     count(array_intersect($placeI,$unWantedI)) != 0
                     &&
                     count(array_intersect($placeJ,$unWantedJ)) != 0)
                          {
                              //Placeholder (nothing needs to happen)
                                The logical was clearer this way
                                with the available php functions
                          }
                          else
                          {
                              for ($k=0; $k<count($oldString); $k++)
                              {
                                  if (trim($oldString[$k]) != "+" &&
                                      trim($oldString[$k]) != "-" &&
                                      $i != $k)
                                  {

                                      $tempPart1 =
                                      trim($oldString[$k]);
                                      $tempParts1 = explode("
                                      ",$tempPart1);

                                      for ($l=0;
                                      $l<count($tempParts1); $l++)
                                      {
                                          if ($tempParts[$j] ==
                                          $tempParts1[$l] && $i
                                          != $k)
                                          {
                                              $tallyMatch++;
                                              $unWantedITemp[] = $k;
                                              $unWantedJTemp[] = $l;
                                              break;
                                          }
                                      }//end L loop
                                  }//end if not + or - for K terms
                                  else
                                  {
                                      if ($i != $k)
                                      {
                                          $tallyMatch++;
                                      }
                                  }
                              }//end k loop
                          }//end if (countintersect)

                          if($tallyMatch == count($oldString)-1)
                          {
                              $newFactor[] = $tempParts[$j];
                              $unWantedI[] = $i;
```

```
97.                             for ($k=0; $k<count($unWantedITemp);
98.                                  $k++)
99.                             {
100.                                $unWantedI[] = $unWantedITemp[$k];
101.                            }
102.                            $unWantedJ[] = $j;
103.                            for ($k=0; $k<count($unWantedJTemp);
104.                                 $k++)
105.                            {
106.                                $unWantedJ[] = $unWantedJTemp[$k];
107.                            }
108.                        }
109.                    }//end j loop
110.                }//end if not + or - for I terms
111.
112.            }//end i loop
113.            $newFactor1 = "";
114.            for ($i=0; $i<count($newFactor); $i++)
115.            {
116.                $newFactor1 .= " ".$newFactor[$i]." ";
117.            }
118.
119.
120.            //*
121.            //Time to rebuild
122.            $newTerms = array();
123.            $term = "";
124.            for ($i=0; $i<count($oldString); $i++)
125.            {
126.                $tempPart = $oldString[$i];
127.                if (trim($tempPart) != "+" && trim($tempPart) != "-
    ")
128.                {
129.                    $tempParts = explode(" ",trim($tempPart));
130.                    for ($j=0; $j<count($tempParts); $j++)
131.                    {
132.                        $omitPart = false;
133.                        for ($k=0; $k<count($unWantedI); $k++)
134.                        {
135.                            if ($unWantedI[$k] == $i &&
136.                                $unWantedJ[$k] == $j)
137.                            {
138.                                $omitPart = true;
139.                            }
140.                        }
141.
142.                        if(!$omitPart)
143.                        {
144.                            $term .= " ".$tempParts[$j]." ";
145.                        }
146.                    }
```

```
147.                     if ($term == "" || $term == " ")
148.                     {
149.                         $term = 1;
150.                     }
151.                     $newTerms[] = $term;
152.                     $term = "";
153.                 }//end if not + or -
154.                 else
155.                 {
156.                     $newTerms[] = $tempPart;
157.                 }
158.             }//end i loop
159.
160.             $newTerms = sortTerms($newTerms);
161.             $newTerms = numeric($newTerms);
162.
163.             if ($newFactor1 != "")
164.             {
165.                 $newString = "(".$newFactor1.")"."(".$newTerms.")";
166.             }
167.             else
168.             {
169.                 $newString = $newTerms;
170.             }
171.             $newString = spacing($newString);
172.         }
173.         else
174.         {
175.             $newString = $oldString;
176.         }
177.
178.         return $newString;
179.
180.     }//endfunction
181.     ?>
```

A.2 Polynomial Factorization

```php
<?php

//This function determines if a plynomial of degree two is
//"cleanly" factorable to sub-polynomials with integer coefficients.
//Higer order polynomial factorzation has been solved but is not
  presented here
//If the input polynomial is not "cleanly" factorable, standard
  factorization  is done.
function factorPolynomial($coeff = array(),$variable = "x")
{
    //function is of the form Ax^2 + Bx + C
        $A = $coeff[0];
        $B = $coeff[1];
        $C = $coeff[2];

        //Generates the prime factorization of each of the integer
          coefficients
        $AFactors = primeFactors($A);
        $CFactors = primeFactors($C);

        //Generates all permutations of the prime factors of each
          number
        $APerms = getFactorsPerms($AFactors);
        $CPerms = getFactorsPerms($CFactors);

        //Resolves each permutation into the desired numerical
          result
        $APermsResolved = resolvePerms($APerms);
        $CPermsResolved = resolvePerms($CPerms);

        foreach($APermsResolved as $AElement)
        {
            $a = $AElement;
            $c = $A/$AElement;

            foreach($CPermsResolved as $CElement)
            {
                $b = $CElement;
                $d = $C/$CElement;

                if (($a*$d + $b*$c) == $B)
                {
                    $coeffs = array($a,$b,$c,$d);

                    $output = "( " . $a . $variable . " + " . $b . "
                    )( " . $c . $variable . " + " . $d . " )";
                    return $output;
                    //return $coeffs;
                }
```

```
48.             }
49.         }
50.
51.         //if we reach here the polynomial is not factorable
52.         return -1;
53.     }
54.
55.     function primeFactors($number)
56.     {
57.         $factors = array();
58.
59.         //resultant set to 1... if not overwritten then the number
60.           is prime
61.         $resultant = 1;
62.
63.         //only need to check up to half
64.         //after that the search would be redundant
65.         $max = $number/2 + 1;
66.         for($i=2; $i<$max; $i++)
67.         {
68.             //% is the modulus operator
69.             //seeing if i evenly divides the number
70.             if ($number%$i == 0 && is_int($number))
71.             {
72.                 //so we store off that factor and see what is left
73.                 //ie the resultant) and then break out of the loop
74.                 //The resultant is now going to be our new "input
75.                   number"
76.                 $factors[] = $i;
77.                 $resultant = $number/$i;
78.                 break;
79.             }
80.         }
81.
82.         //If the resultant is not equal to 1 then our input number
83.         //was not prime and we need to obtain the prime
84.         //factorization for that resultany
85.
86.         if($resultant != 1)
87.         {
88.             //Here we call the function primeFactors with the
89.               function primeFactors
90.             //This is known as recursion and is very helpful for
91.               prolbems of this nature
92.             $temp = primeFactors($resultant);
93.             $factors = array_merge($factors,$temp);
94.         }
95.         else
96.         {
97.             //If our resultant was 1 then our previous number had no
98.               factors
```

```
99.             //meaning it was prime.  So we store it off as the last
100.              factor and
101.            //we exit whatever level of recursion we are on.
102.            $factors[] = $number;
103.        }
104.
105.        return $factors;
106.    }
107.
108.    //For any list of factors this function will produce all of the
109.    //permutations of that list
110.    function getFactorsPerms($factors)
111.    {
112.        $perms = array();
113.
114.        if (count($factors) == 1)
115.        {
116.            $perms[] = $factors[0];
117.            return $perms;
118.        }
119.
120.        $prefix = array(array_shift($factors));
121.
122.        $suffix = $factors;
123.        $suffix = getFactorsPerms($suffix);
124.
125.        $perms[] = $prefix[0];
126.        $addperm = $prefix[0];
127.        for ($i=0; $i<count($suffix); $i++)
128.        {
129.            $perm = $prefix[0] . " " . $suffix[$i];
130.            $perms[] = $perm;
131.        }
132.
133.        $perms = array_merge($perms,$suffix);
134.
135.        return $perms;
136.    }
137.
138.    //For any each of the permuations we need the actual number
139.    //which is the multiplication of those
140.    function resolvePerms($perms)
141.    {
142.        $resolved = array();
143.        foreach ($perms as $perm)
144.        {
145.            $primes = explode(" ",$perm);
146.
147.            $coeff = 1;
148.            foreach ($primes as $prime)
149.            {
```

```
150.                 $coeff *= $prime;
151.             }
152.              $resolved[] = $coeff;
153.         }
154.          return $resolved;
155.     }
156.
157.     ?>
```