A Problem Generator to Learn Expression Evaluation in CS I, And its Effectiveness
Amruth N Kumar and Aravind K Krishna
Ramapo College of New Jersey
505 Ramapo Valley Road
Mahwah, NJ 07430-1680
amruth@ramapo.edu, ak_krishna@yahoo.com

Abstract
Introductory Computer Science students often do not clearly understand the concepts of operator precedence and associativity, and make mistakes when they write and evaluate expressions. In order to help them learn better by solving problems, we have developed an applet that generates problems (henceforth called problet) and corrects the learner’s answers. In this paper, we will describe the design and implementation of this problet.

Whereas problem-solving is known to improve learning, it is not clear that using online resources such as problets to practice problem-solving is quite as effective. Therefore, we conducted a controlled experiment on the effectiveness of using our problet. Results indicate that using the problet did improve learning by 18%. We will describe the experiment and present its results.

1 Introduction
Problem-based learning improves long-term retention [7], and is better than traditional instruction for improving the ability of students to solve real-life problems. Textbooks are generally inadequate as sources of problems because of their limited, non-interactive nature.

Moreover, textbooks do not or cannot present the steps required to solve a problem, the visualizations that are helpful in solving the problem, or the feedback that helps a student learn from his/her mistakes. Even in disciplines such as Physics and Mathematics, where textbooks generally tend to include many more practice problems than in Computer Science, faculty are increasingly turning to the use of technology to address this issue. Typically, programs are being written to generate problems, and such programs are being made available to students for practice, e.g., CAPA [8] for physics, and CHARLIE [3] for electronics and control systems.

In Physics, the use of problem generation systems has been shown to increase student performance by 10% [9], largely due to increased time spent on the task. We wanted to check whether we could reproduce this result in Computer Science using our problet. Therefore, we conducted a controlled experiment. We will describe our experiment, and its results later in this paper.

2 Problem Domain
Surprisingly, introductory students find the concepts of operator precedence and associativity hard to understand and use. Our experience indicates that students make numerous mistakes both when writing expressions and analyzing them in programs. While students can get around this problem by generously using parentheses in expressions, this is not a substitute for understanding operator precedence and associativity.
We have found that an effective strategy to practice precedence and associativity is for students to use underbraces when they evaluate expressions: at each step, they draw an underbrace to span the next sub-expression being evaluated, and write the partial result of evaluating the sub-expression under the underbrace. We have used this very same strategy in our problet: the problet uses underbraces to indicate the order in which operators are evaluated in an expression, and users are expected to click and drag the mouse to draw underbraces when evaluating expressions. Therefore, the problet is capable of verifying not only the correctness of the final answer, but also the correctness of the order of evaluation of operators, and intermediate results too.

3 Problet Design

3.1 Expression Generation

The problet generates an expression as a binary tree. The terminal nodes of the tree contain operands and non-terminal nodes contain operators. Associated with each instance of an operator are its precedence, associativity and arity.

The tree generation algorithm is recursive. It randomly designates the nodes to be either operators or operands. Operands are generated using a random number generator. Operators are selected randomly from among the operator types selected by the user. Certain restrictions are imposed on operator nodes during tree generation to ensure the validity of the generated expression. If the current node is a left-associative operator, then:

a) The precedence of the left child node is guaranteed to be greater or equal to the precedence of the current node.

b) The precedence of the right child node is guaranteed to be greater than the precedence of the current node.

For right-associative operators, the above rules are inverted. Right-associative unary operator nodes have only the right child node, and vice versa. Finally, the left child node of an assignment operator must be a variable.

Heuristics are used to improve the quality of the generated expressions:

- The root node of the binary tree is selected from among the operators with the lowest precedence. This increases the chances of the generated problem containing operators with different precedence levels.

- The operands are randomly generated in the range 1-20, so that the resulting expression is easy for users to evaluate mentally. A calculator is also included in the problet to help users with calculations. But, our intention is to make its use as unnecessary as possible, since the problet is designed to teach and test operator precedence and associativity, not the user’s ability to do mental math.

- The probability of a node being an operand increases with the depth of the node in the tree. This curtails the generation of long expressions with too many operators.

The tree is traversed in in-order fashion to produce the infix expression displayed to the user. The expression is evaluated by traversing a copy of the tree in post-order fashion. The intermediate results are stored in the nodes of the tree and used to check the user’s answer. The x-coordinates of the operands are also stored in the nodes, and are used to verify the underbraces in the user’s answer.

3.2 User Interface

The problet allows the user to choose from various operator types namely, logical, arithmetic, relational, bit-wise and assignment in C++. The user may choose either a single operator type or multiple operator types. The problet displays a table of the precedence and associativity of the operators of the selected types as shown in Figure 1. The user has the option to generate problems at three different levels of hardness. At the basic level of hardness, the expression contains less than 3 operators, at the intermediate level it contains from 3 to 5 operators, and at the advanced level it contains from 5 to 8 operators. The default is intermediate level of hardness.

![Problet showing table of operator precedence and associativity.](image)

The problet uses a new window to display the problem generated based on user’s preference of operators and hardness. The user is expected to evaluate the expression
by drawing underbraces and writing down intermediate results. In order to draw an underbrace, say under a binary expression, the user must click the mouse under the left operand, drag it across the sub-expression, and release it under the right operand. When the user draws an underbrace, an input box pops up for the user to enter the partial result as shown in Figure 2.

Figure 2. Problem Window, where the user evaluates the expression.

If the user wants to modify a partial result entered earlier, or erase an underbrace already drawn, the user may double-click on the corresponding underbrace. This pops up a dialog box with options to do either of the above tasks.

Finally, as noted earlier, the user has access to a calculator to help him/her with laborious numeric calculations when solving problems. At this point, the user may enter either the tutorial mode or the test mode to solve the presented problem.

3.3 Tutorial/Analysis Mode

The user may enter the tutorial mode by repeatedly clicking on the “Step Through” button. On each click, the prolet displays an underbrace below the next sub-expression to be evaluated, followed by the result of the evaluation. Therefore, the user can observe the sequence in which the expression is evaluated, and learn the correct order of operator evaluation.

3.4 Test Mode and Feedback

The user may enter the test mode by simply solving the problem online and clicking on the “Check My Answer” button. In order to solve the problem, the user must draw underbraces and enter intermediate results as described before. The user may have his/her answer checked at any stage of solving the problem.

The prolet provides the following feedback to the user, as shown in Figure 3:

a. On a separate screen, it graphically displays the correct order of evaluation of operators in the expression, along with all the partial results, so that the user can compare his/her answer with the correct answer.

b. Next, it lists the number of operators that the user evaluated in the correct order, followed by the number of correct partial results for the operators evaluated in the correct order.

The user may generate a fresh problem by clicking on the “New Problem” button, whereupon, the feedback window is automatically closed and a new problem is displayed in the problem window.

Figure 3. Feedback Window.

3.5 Implementation Details

The prolet is implemented in Java (Version JDK 1.2.1), and uses Swing classes. It consists of 7 classes, and is about 33K in size.

Evaluating the Expression: Initially, we were storing the expression in in-fix form as a string. We attempted to evaluate the expression by first converting it to postfix notation, and using a stack data structure to evaluate the postfix equivalent. Although this gave us the correct final answer, the order of evaluation of intermediate answers was not correct according to operator associativity.

We later switched to generating and storing the expression as a binary tree. We traversed the tree in post-order fashion, calculating the values of sub-expressions, and storing them in the nodes of the tree. However, we could not use in-order traversal of the tree to illustrate the step by step evaluation of the expression using underbraces because, in
certain cases, this violated operator precedence rules. E.g., consider the expression represented by the binary tree shown in Figure 4. An in-order traversal of the tree would display the result of 3 < 4 before the result of 8 * 9 which violates precedence rules. To solve this problem, we stored the intermediate results along with the intermediate expressions in an array. We sorted the array according to the precedence of the operators. The sorted array now has the correct order of evaluation, and preserves the associativity of operators. We used this sorted array to check the user’s answer and to display the correct answer to the user.

Figure 4. Binary tree representation of an expression

Data Structure for Operators: We use a vector of vector objects to hold information about operators. Each index in the vector corresponds to one level of precedence, and the vector at that index contains the various operators at that level of precedence. This organization makes it easy for us to add new operators to the problet.

4 Effectiveness of Using Problets

We conducted a controlled experiment to assess the effectiveness of using our problet to learn problem-solving. The test subjects were students in the Upward Bound project, who were junior and senior students from area high schools taking their first programming course. The protocol for the experiment was as follows:

Lecture: In the first ten minutes, all the students attended a common lecture on precedence, associativity, integer division and remainder operators in C++. Students were introduced to the practice of using underbraces to indicate the order in which an expression is evaluated, and noting the intermediate result under each underbrace.

PreTest: Next, all the students answered a written 10-minute pre-test which consisted of 10 questions. Students were told that the test would measure the effectiveness of the lecture and they were urged to do their best on the test. They were reminded to use underbraces and write down intermediate results when solving problems.

Practice: Students next engaged in a practice session. First, students were divided into test and control groups. The students in the test group used the problet for practice, each being seated at a separate computer. The students in the control group used printed workbooks for practice, each being provided their own copy.

Post-Test: Finally, the students from both the control and test groups sat together and answered a written 10-minute post-test that consisted of 10 questions. Students were told that the post-test would measure the effect of practice on their learning, and they were urged to do their best on the test.

Follow-Up Test: A week later, students from both the control and test groups sat together and answered a written 10-minute follow-up test. The test consisted of 5 questions each from the pretest and post-test. The 5 questions selected from each test included 2 that the most students had answered correctly, 2 that the fewest students had answered correctly, and an arbitrarily chosen question. Neither group was privy to the answers for the pretest or the post-test before they took the follow-up test. Neither group had access to the tests themselves during the intervening week. Since the problems had been randomly generated, we did not expect that any student would remember the questions, let alone their answers from the previous week.

The questions in the pretest, post-test, and printed workbook were all expressions consisting of 4 operators. The instructor had generated all the questions using the problet, and later, formatted and printed them on paper. Our observations are summarized in Table 1. The table lists the averages and standard deviations of the pretest, post-test and follow-up test scores, and the number of questions attempted by students during the practice session, first for the test group which used the problet (9 students), and then for the control group (13 students). The figures were calculated after eliminating outliers (3 students). Test scores are out of 40.

Table 1. Average Scores of Test and Control Group

<table>
<thead>
<tr>
<th></th>
<th>Pre-Test Score</th>
<th>Problems Solved</th>
<th>Post-Test Score</th>
<th>Follow-Up Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Avg.</td>
<td>28.56</td>
<td>13.89</td>
<td>33.78</td>
<td>35.13</td>
</tr>
<tr>
<td>Std-Dev</td>
<td>9.02</td>
<td>4.14</td>
<td>7.95</td>
<td>8.08</td>
</tr>
<tr>
<td>Control Avg.</td>
<td>25.92</td>
<td>16.23</td>
<td>34.38</td>
<td>34.67</td>
</tr>
<tr>
<td>Std-Dev</td>
<td>7.05</td>
<td>3.32</td>
<td>7.15</td>
<td>6.97</td>
</tr>
</tbody>
</table>

Observations: The average score of the test group, which used the problet improved from 28.56 on the pretest to 33.78 on the post-test, which represents an 18.27% improvement. Therefore, using the problet did help students learn problem solving through practice.
However, this improvement was not as high as that for the control group which used the printed workbook (33.02%). In other words, students who practiced using traditional printed workbooks seemed to have learned better than those who used online problets.

One explanation for this discrepancy may be that the test group students took longer to get used to the technology of the problet. We observed that some students had questions about the technology even after our introduction, e.g., clicking and dragging to draw underbraces, double-clicking to change an intermediate result, etc. We had to intervene and give individual instruction to some students. Since the time allotted for practice was limited (10 minutes), this process of getting used to the technology cut into the time that students spent learning problem solving.

This explanation is also borne out by the fact that the average number of practice problems solved by the control group students who used the printed workbook (16.23) is 16.84% higher than the average number of practice problems solved by the test group students who used the online problet (13.89). The control group average could have been even higher - the printed workbooks contained only 20 questions, and a few students who used the workbook finished the practice session early.

The test and control groups were both introduced to real arithmetic in the days between their post-test and the follow-up test given a week later. They also practiced precedence and associativity with both integer and real operands. From the average scores of the two groups on the follow-up test we note that the test group score continued to improve during this time, whereas the control group score stayed the same. We conclude that any handicap that the test group may have suffered from using online problets for practice was not long lasting.

We used $t$-test and randomization test [6] to compare the performance of the test and control groups. Both the tests indicated that there was no systematic difference between the test and control groups, i.e., the difference in their performance on the pretest, post-test and follow-up test could all be attributed to pure chance. So, the use of problets may have been just as effective as using printed workbooks for practice.

We observed a ceiling effect on the tests, wherein anywhere from 7% to 50% of the students scored the maximum on the tests. This may have obscured some systematic difference between the test and control groups. We intend to correct for this in the future.

Finally, there are many advantages to using problets instead of workbooks for practice. It is labor-intensive to generate the printed workbook with questions and their answers. In contrast, the problet generates problems as well as their solutions (and even grades user’s answers) automatically. Unlike the workbook, the problet provides an unlimited supply of problems. Since it generates problems randomly, the problems are endlessly individualized, i.e., each student gets his/her own set of problems, unlike in the case of the printed workbook. This individualization helps prevent cheating on tests and assignments.

References