A TUTOR FOR USING DYNAMIC MEMORY IN C++
Amruth N. Kumar

Abstract — Using dynamic memory has been a source of frustration for Computer Science and Engineering students programming in C++. We designed a tutor to help students understand dynamic memory and avoid the pitfalls of using it. The tutor automatically generates problems on dynamic memory usage, and provides detailed feedback on the user’s answers. Our evaluation of the tutor indicates that using it helps improve both the cognitive and affective learning of students. In this paper, we will describe the design of the tutor, discuss our evaluation protocol and present the results of evaluation.

Index Terms — Evaluation of Educational Software, C++ Programming, Technology for Education, Tutoring System.

INTRODUCTION
Using dynamic memory has been a source of frustration for Computer Science and Engineering students programming in C++. Dynamic memory (heap) is recommended for the implementation of many data structures (e.g., linked lists, queues), so learning the syntax and semantics of using dynamic memory is essential to most large-scale C++ programming.

Our experience indicates that there are three reasons why students find it hard to work with dynamic memory:
• Memory leakage: students forget to deallocate dynamic memory that is no longer being used.
• Illegal memory accesses through dangling pointers: students attempt to access deallocated memory through pointers, or access memory through unassigned pointers.
• The notion that heap objects do not have an independent scope other than that of the pointer pointing to them.

We designed a tutor to help students understand dynamic memory and avoid these pitfalls. The tutor engages the student learner in active learning through problem-solving. It dynamically generates C++ code segments and asks the learner to identify whether and what errors the code may contain. This feedback includes identifying and explaining any memory leakage, dangling pointers, and scope issues.

We have tested this tutor in several sections of our Computer Science II course since Spring 2001, and assessed its effect on both cognitive and affective learning of our students. We have observed a significant improvement in student scores from the pretest to the post-test when students used the tutor to practice solving problems between the tests.

On feedback forms, a majority of the students also noted that the tutor helped them better learn the material.

In this paper, we will describe the design and user interface of our tutor. We will discuss the protocol and results of evaluating the tutor. We have presented a related tutor on pointers for indirect addressing elsewhere [13]. This paper builds upon the earlier work by extending the domain of the tutor to include dynamic memory, global variables and multiple functions; the paper introduces multiple forms of feedback, and presents new results of evaluating the tutor in our classes. The tutor is designed to be used as a supplement to classroom instruction, either in a closed laboratory or for after-class assignments. It targets application in Bloom’s taxonomy [5].

THE DESIGN OF THE TUTOR
We will describe the tutor in terms of the following features:
• Problem Generation: The tutor automatically generates problems, and is capable of generating an unlimited number of problems.
• User Interface: The tutor provides a clock-wise interface that is easy to use.
• Feedback: The tutor grades the user’s answer and provides detailed feedback about the correct answer.

Problem Generation
The problems generated by the tutor consist of a C++ program and a set of input options for errors that may be found in the program. The tutor generates C++ programs as randomized instances of pre-defined parameterized problem templates. Since the tutor has more than 40 templates, and each template can be infinitely randomized, the tutor is capable of generating a combinatorially explosive number of un-identical programs/problems, and the user may rarely, if ever see the same problem twice. Some advantages of such problem generation are:
• The user may use the tutor to practice solving problems as long as (s)he pleases.
• The tutor may be used for assignments without concern for plagiarism, and for tests without concern for cheating.

The templates are specified in pseudo-BNF notation. For example, consider the following template:

```cpp
{{<T1><V1>=<R#>;<T1><P1>;<P1>=new<R#>;<<<P1>;<<<V1>;
```
In the template, T refers to a data type, V a variable, R# a random number, P a pointer, and << refers to an output statement. Braces {} indicate scope objects, the outermost pair corresponding to the program, and the nested braces corresponding to functions and blocks.

The C++ programs generated by the tutor may contain one or more functions, pointers and local and global variables. The tutor randomly introduces distracter statements into the code to make the problems more interesting. The tutor focuses on semantic (logical) and run-time errors in the code. Therefore, it does not generate programs with obvious syntax errors such as a variable being referenced before it is declared.

The user is asked to identify whether the program contains any dangling pointers, lost objects, syntax errors (such as a variable being referenced outside its scope), semantic errors (such as referencing the value of an uninitialized variable), or run-time errors. A program may contain more than one error, e.g., both a dangling pointer and a lost object. The user also has the option of indicating that the code is correct.

User Interface

The user interface of the tutor consists of the following components:

- **Problem Panel**, where the C++ program is displayed;
- **Answer Panel**, where the question about the program is posed, and input options are provided for the user to enter his/her answer(s) – the user may choose more than one answer for a problem;
- **Feedback Panel**, where feedback about the user’s answer is displayed;
- **Statistics Panel**, where usage statistics are displayed for the user and session: the number of problems attempted, and the number of problems solved correctly, incorrectly, and partially.
- **Menu Options** to choose among feedback types, formatting for the program and feedback, etc.

Figure 1 shows the user interface, consisting of the above components. The user is led through a clockwise flow of action: from the problem panel in the top left, to the answer panel in the top right, to the “Check My Answer” button in the right center (not shown in the figure), to the feedback panel and the “Next Problem” button in the bottom right, to the statistics panel in the bottom left, and back to the problem panel in the top left for the next problem. The tutor makes the “Check My Answer” and “Next Problem” buttons available only in their correct contexts to avoid confusion.

Types of Feedback

When the user clicks on “Check My Answer” after entering his/her answer, the tutor states whether the answer is correct, incomplete or wrong. In addition, it provides feedback explaining the correct answer. The tutor currently provides three types of feedback [1]:

- **Demand Feedback** – The tutor provides feedback only when the user requests. The feedback includes whether the user’s answer is correct and an explanation for the correct answer. The explanation is very detailed, and includes a line-by-line commentary of what the code does and where the code contains errors.
- **Error-Flag** – As soon as the user selects an answer, the tutor signals whether the answer is correct or incorrect by color-coding the selection, red for incorrect and green for correct. The user may follow-up by asking for feedback, or select additional answers.
- **Immediate Feedback** – If the answer selected by the user is incorrect, in addition to color-coding the selection to indicate that the answer is incorrect, the tutor guides the user through three progressively descriptive levels of hints: **abstract** (e.g., “Remember, a heap object is lost if the pointer pointing to it is reassigned and no other pointer is pointing to it”), **concrete** (e.g., “Was the value of indirectPointer assigned to any other pointer before indirectPointer was re-assigned?”) and **bottom-out** (e.g., “Well, referencePointer was assigned the value of indirectPointer before indirectPointer was re-assigned. So, the heap object originally pointed to by indirectPointer is still accessible through referencePointer, and is not lost.”).

To summarize, in demand feedback the tutor corrects and explains on request; in error-flagging, it corrects without being prompted, but explains on request; and in immediate feedback, it both corrects and explains without being prompted.

FIGURE 1
SNAPSHOT OF THE SOFTWARE, WITH FEEDBACK
EVALUATION OF THE TUTOR

We have been evaluating various versions of this tutor since Fall 2000 in the Computer Science II course. We began conducting controlled tests of the tutor in Spring 2001, and started evaluating the tutor for dynamic memory usage in Fall 2001.

Our long-term objective has been to evaluate whether using such tutors for problem-solving helps improve the learning of students. Currently, exercises in textbooks are the alternative most widely available to students for learning by solving problems. Two of the ways in which our tutor is different from exercises in textbooks are:

1. The problems generated by the tutor are delivered electronically, whereas the textbook exercises are delivered in hardcopy; Solving problems generated by the tutor requires active interaction with the tutor;
2. The tutor provides feedback, i.e., explanation in addition to the correct answer, whereas most textbooks list only the correct answer;

To address the first difference, in Spring 2001, we compared using the tutor against using a printed workbook for practice. To address the second difference, in Fall 2001, we compared two versions of demand feedback in the tutor – minimal, where no explanation is given for the correct answer, versus full, where a detailed explanation is given for the correct answer.

In this section, we will first describe the protocol we used for the tests, and then discuss the results we obtained from the tests. We will present results pertaining to both cognitive and affective learning. All the Computer Science II class sections where the tutor was evaluated had the same instructor, who was not part of the development team of the tutor.

The Protocol

The protocol for the tests was as follows:

PreTest: The students answered a written pretest in 8 minutes. Students were told that the pretest assessed how much they had learned from classroom lectures. They were urged to correctly solve as many problems as they could.

Practice: Students were randomly divided into control and test groups. Each student was seated at a separate computer, with the tutor already launched. Students were shown how to use their instrument of practice (printed workbook and tutor in Spring 2001, tutor with minimal feedback and tutor with full feedback in Fall 2001). Students were then asked to solve as many problems as they could in the next 12 minutes, using the instrument of practice assigned to them. Where applicable, students were asked to pay special attention to the feedback provided by the tutor.

Post-Test: The students answered a written post-test in 8 minutes. Students were told that the post-test assessed how much (more) they had learned during the practice session. Again, they were urged to correctly solve as many problems as they could in the given time.

Feedback: Finally, students were asked to fill out a feedback form about their impressions of the instrument of practice they had used.

Note that the questions on the pretest, the post-test, and the printed workbook (Spring 2001) had themselves been generated earlier using the tutor. In the printed workbook, answers to all the questions (but not explanations) were listed at the end of the workbook. The same answering options were provided on the pretest, the post-test, and the printed workbook as in the tutor.

Evaluation Results: Cognitive Learning

We compared using the tutor versus using a printed workbook for practice in Spring 2001. Table 1 lists the average score and standard deviation for the test group which used the tutor and the control group which used the workbook (N=33, 16 in Test and 17 in Control).

<table>
<thead>
<tr>
<th>Test Group</th>
<th>PreTest</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>13.00</td>
<td>23.06</td>
</tr>
<tr>
<td>Standard-Dev</td>
<td>6.61</td>
<td>10.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Group</th>
<th>PreTest</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>15.24</td>
<td>24.71</td>
</tr>
<tr>
<td>Standard-Dev</td>
<td>7.10</td>
<td>10.54</td>
</tr>
</tbody>
</table>

The figures in the table indicate that the improvement in the test group was marginally better than that in control group, although the difference is not statistically significant. The figures however confirm that practicing problem-solving helps improve learning. Given this result, it is easy to argue the merits of using an automated tutor for practicing problem-solving rather than a printed workbook – the tutor can generate an unlimited number of problems, it can provide instantaneous feedback, and it can record student usage/performance.

In Fall 2001, we compared two versions of demand feedback in the tutor – minimal, where no explanation is given for the correct answer, versus full, where a detailed explanation is given for the correct answer. Table 2 lists the average score and standard deviation for the test group which got full feedback and the control group which got minimal feedback (N=22, 14 in Test and 8 in Control).

<table>
<thead>
<tr>
<th>Test Group</th>
<th>PreTest</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>11.57</td>
<td>27.29</td>
</tr>
<tr>
<td>Standard-Dev</td>
<td>7.91</td>
<td>21.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Group</th>
<th>PreTest</th>
<th>Post-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>8.25</td>
<td>17.38</td>
</tr>
<tr>
<td>Standard-Dev</td>
<td>5.60</td>
<td>14.62</td>
</tr>
</tbody>
</table>
The figures in the table indicate that the improvement in the test group was better than that in the control group. In Spring 2002, we compared the average points per attempted question for the control and test groups. Table 3 lists the average points per question, 2 being the maximum possible points per question (N=16, 8 in Test and 8 in Control).

### Table III

**SPRING 2002: MINIMAL VS MAXIMAL FEEDBACK IN THE TUTOR**

<table>
<thead>
<tr>
<th></th>
<th>PreTest</th>
<th>Post-Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Group</strong></td>
<td>0.62</td>
<td>1.02</td>
</tr>
<tr>
<td><strong>Control Group</strong></td>
<td>0.63</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Once again, the improvement in the test group was better than that in the control group. However, whether minimal feedback is provided or full feedback, the learning of the students seems to considerably improve after using the tutor for practice. Clearly, tutors such as this have a role to play in Computer Science education. We plan to continue to evaluate the tutor in future semesters.

**Evaluation Results: Affective Learning**

We analyzed the feedback forms filled out by the students in the last step of the protocol, to evaluate the usability, learnability and usefulness of the tutor. Items on the feedback form also provided insight into affective learning of the students: whether and how much they thought they had learned by using the tutor.

In Spring 2001, when we compared the tutor with printed workbooks, students were asked to provide feedback on the “instrument” they had used for practice: tutor for test group and workbook for control group. The test versus control averages of student feedback on a Likert scale of 1 (Strongly Agree) to 5 (Strongly Disagree) on selected questions are given below:

- The instrument provided useful feedback: 2.20 vs 3.06.
- The instrument helped me learn the material: 2.31 vs 2.88.
- This instrument should be made available to all the students: 1.56 vs 2.65.
- If this instrument is made available, I will use it: 1.93 vs 2.65.
- I would like to see such instruments on other topics: 1.44 vs 2.59.

Clearly, students who had used the tutor were more positive about the feedback they had received, and how much they had learned, than those who had used the printed workbook. They were also much more enthusiastic about re-using their practice instrument (i.e., tutor for test group), making the instrument available to other students, and accessing similar instruments on other topics.

Some of the comments written by students who used the printed workbook, that support the preferability of tutors include: “These (booklets) should also be available online so students can use them whenever”, and “I found the booklet to be no help whatsoever as it only provided answers and no explanation was given”. This latter sentiment was expressed by at least 7 other students in their comments. Some of the comments written by students who used the tutor include: “Learning complex things in this format can be much less intimidating than seeing a whole programming example in a book”, and “Instantaneous feedback is a very useful and under-used tool. This program allows students to get that needed feedback with little effort.”

Recall that in Fall 2001, we had compared full feedback (provided to test group) versus minimal feedback (provided to control group). The test versus control averages of student feedback on a Likert scale of 1 (Strongly Agree) to 5 (Strongly Disagree) on selected questions are given below:

- The feedback provided to my answers was useful: 2.57 vs 2.83.
- The feedback provided to my answers was sufficient: 2.36 vs 2.67.
- The tutor helped me better understand what I knew: 2.50 vs 2.83.
- The tutor helped me learn new material: 2.36 vs 3.00.

Clearly, students who had received full feedback were more positive about the feedback they had received, and how much they had learned. For the statement “It was easy to learn to use this tutor”, the combined average of the test and control groups was 1.63, which validates the design of the user interface of the tutor.

Some of the comments written by students who got minimal feedback, that support the provision of full feedback include: “It should tell you why you got the answer wrong and explain it, other than that I found it could be a helpful tool”, and “I would have liked to know what was wrong with the code besides saying what error, like show why it is the way it is”.

**RELATED WORK**

Problem-based learning improves long-term retention [9]. A tutoring system such as ours has several advantages over textbooks, the traditional source of problems for students:

- Unlike textbooks, the tutor can generate an unlimited supply of problems, thereby providing as much practice with problem-solving as the learner wants.
- Unlike textbooks, the tutor can instantaneously grade the user’s answer and provide feedback.
- Unlike textbooks, the tutor can provide detailed feedback.

Therefore, our tutor may be used in many ways in a course:

- Instructors may use our tutor to assign homework without the fear of plagiarism and administer tests without concern for cheating.
- Instructors may use our tutor to promote active learning in class, and in distance education courses.

In Computer Science, various researchers have advocated the use of self-paced exercises [15], practice to build problem-solving skills [17], and the use of frequent, graded
assignments in a course [8]. Our tutor addresses all these concerns.

Tutoring systems such as ours have been developed for quantitative disciplines such as Physics (e.g., CAPA [10]), and electronics and control systems (e.g., CHARLIE [4]). Examples of such tutoring systems developed for Computer Science include PILOT [6], SAIL [7] and Gateway labs [3]. PILOT is a problem generation tool for graph algorithms, SAIL is a LaTex-based scripting tool for problem generation, and Gateway Labs generate problems on mathematical foundations of Computer Science. In our work, we have attempted to build tutoring systems for problems based on programs, problems for which the answers may not always be quantitative [13,14,16,18]. WebToTeach [2] has an objective similar to our tutor, but it administers problems created by the instructor, and does not itself generate the problems or explain their solution.

The use of problem generation systems has been shown to increase student performance by 10% in Physics [11], largely due to increased time spent on the task. Our evaluations support this result, indicating that tutoring systems do have a role to play in Computer Science higher education.

**FUTURE WORK**

We have developed tutors for other selected topics in Computer Science, including expression evaluation in C++ [12], pointers for indirect addressing in C++ [13], nested selection statements in C++ [16] and parameter passing in programming languages[18]. Our long-term goal is to abstract out the common components among these tutors, and develop a reusable framework for the implementation of tutors. We need to gain more experience in building tutors before we can develop such a framework.

We plan to continue to test our tutor in future sections of Computer Science II course. We also plan to add additional templates to the tutor involving storage classes, function return types and structures in C++.

The tutor is implemented as a Java applet so that it can be accessed over the Web without constraints of time and space. The tutor uses Swing classes, consists of 30 classes, and is about 640K in size. It is currently available over the Web at http://orion.ramapo.edu/~amruth/problems

**ACKNOWLEDGMENT**

Partial support for this work was provided by the National Science Foundation’s Course, Curriculum and Laboratory Improvement Program under grant DUE-0088864.

**REFERENCES**


