Using Java to Help Students Practice Problem-Solving

Article · January 2000

CITATIONS
2

READS
4,663

2 authors, including:

Amruth Kumar
Ramapo College
183 PUBLICATIONS **1,388** CITATIONS

SEE PROFILE

Using Java to Help Students Practice Problem-Solving

Amruth N Kumar, Neeraj Singhal
Ramapo College of New Jersey
505 Ramapo Valley Road
Mahwah, NJ 07430-1680
amruth@ramapo.edu, Neerajs@att.net

Abstract

In many disciplines in Science, including Computer Science, solving problems is an essential part of learning. Solving problems not only helps improve retention (Farnsworth, 1994), but also provides the practice necessary for students to be able to apply their learning.

In this paper, we report on the use of Java applets to automatically generate problems on a given topic. Such applets may be used by students to reinforce their learning by solving problems online. The applets may be used as supplements to textbooks, for administering tests online, and as online resources in distance learning courses.

We refer to the Java applets built to automatically generate problems, as problets. In the paper, we will discuss the characteristics and components of problets, and present the design of two problets that we have implemented for static scope in Pascal and nested selection statements in C++.

1. Introduction

Solving problems is an inextricable part of learning the Sciences such as Computer Science. The more the problems solved by the learner, the better the understanding of the concept. Problem solving is essential not only to thorough understanding of a concept, but also to its subsequent application to real life situations.

To solve more and varied types of problems, a student must have access to such problems. The most common source of problems is generally the textbook. However, as a source of problems, textbooks are limited, static, non-interactive and lack animation.

1. **Limited**: The selection of problems in a textbook is limited because of space constraints, and is generally similar in nature across various textbooks. Once solutions are made available for these problems (either by the instructor or by a solutions manual), they cannot be used any longer for testing or assignments.

2. **Static**: The limited nature of problems in a textbook is compounded by the fact that they are the same problems available across editions and schools. They cannot be parameterized from one learner to the next, and do not take into account the level of expertise of the learner.
3. **Non-interactive**: When a student solves a problem, the textbooks are not capable of providing feedback on whether and why the student's answers are correct/wrong, feedback that helps a student learn from his/her mistakes.
4. **No animation**: Textbooks seldom include animations and visualizations that are helpful in solving a problem, such as alternative representations of the problem space, animation of the procedure to solve a problem, etc.

In order to address these problems, we propose Java applets that automatically generate problems, provide necessary visualizations and animations, solve problems, and provide feedback to the user. We refer to such applets designed to generate problems as problets. In the next section, we will discuss the suitability of Java for implementing problets. In Section 3, we will list all the ways in which problets can be used for teaching and learning. We will discuss the minimal capabilities of problets in Section 4. We will list the components of a problet in Section 5, and mention some issues that must be addressed when developing a problet. In the two sections after that, we will present the design of problets for static scope in Pascal and nested selection statements in C++. We conclude with future directions for this work.

2. Java for Problets

Java is ideal for implementing problets because of several rather obvious reasons:

- **Non-proprietary, Platform-independent and Free**: Java is not proprietary. Programs written in Java are platform-independent, and the tools needed to build (JDK) and run Java programs (browsers) are available for free. Therefore, students who want to run problets on their home computers do not have to buy additional/expensive software/hardware. Instructors who want to build a problet do not have to port it to different platforms, which saves them considerable amount of time and aggravation.
- **Ease of Use**: Running Java problets is as simple as launching them inside a Web browser. Students do not have to explicitly download any programs, and go through the trouble of installing them on their computer. Therefore, using problets could not be any easier.
- **Any time, Anywhere Accessibility**: Implementation in Java permits Web-based delivery of problets. The obvious advantages of Web-based delivery are that the problets will be available any time, not just during class or office hours, and anywhere, not just the school laboratories.
- **Easy GUI**: Before the advent of Java, providing a Graphical User Interface for an application meant the implementor had to write it in proprietary software that was not always portable across platforms. Java provides facilities for construction of Interfaces that are not only portable but also free. With Swing, Java provides the same look and feel, immaterial of the underlying hardware, and hence, true platform-independence.

  Java has made building problets cheaper in terms of time, free in terms of deployment, and universally accessible around the clock.

3. Uses of Problets

Problets may be used in several ways in a course :

1. **Student Practice**: Problem-based learning helps to clarify concepts and helps the student better understand the subject. It is known to improve long-term retention (Farnsworth, 1994), and is better than traditional instruction for improving the ability of students to solve real-life problems. However, this method of education is time consuming both inside and outside the classroom (Vernon, 1995). Problets may be used in a course to reduce the time spent inside the classroom on problem-based learning: problets may be made available to the students outside the classroom, and students may be directed to solve a certain number of problems as part of the course.
2. **Designing Test and Assignment Problems**: Problets can be used by instructors to set problems for tests and assignments in a course. Problets are especially useful for instructors who teach the same course year after year and are reluctant to use the same set of problems from one semester to another. They may use the problets to provide a fresh set of validated problems to the students every semester. By using problets to automatically generate problems, instructors can save considerable amount of time generating problems, validating them, and solving them, all of which are time-consuming tasks.
3. **Administering Online Tests**: Instructors can use problets to conduct on-line tests that involve problem solving. By randomizing the problems generated by an applet, and serving a different problem instance to each student, instructors can thwart cheating during the test. Students can be provided instant feedback on how well they did on the test. Students may also be given the chance to answer as many questions as they please during the test, until they obtain a score satisfactory to them. Finally, by making the problets available to students to prepare for the test, and thereby, giving them an opportunity to familiarize themselves with the testing environment, instructors can reduce students' stress associated with taking the test.
4. **Distance Education**: Problets may be used as an integral component of distance learning courses, wherein, active online learning is critical to the success of a program. Instructors may make several problets available online each week, and require that their students solve a minimum number of problems on each problet. Instructors may automatically monitor their students' efforts in the course by collecting usage information from the problets.

4. Capabilities of Problets

We have identified the following as the minimal capabilities a typical problet must possess:

- **Automatic Problem Generation**: A problet must be capable of generating problems *automatically*. In addition, the problet should preferably not generate the same problem twice during a session. Systems that generate problems have been discussed in Intelligent Tutoring Systems community for at least two decades (e.g., Brusilovsky, 1993, Koffman & Blount, 1975, Wenger, 1987). In such systems, problems may be either generated dynamically, or

merely selected from a compiled repertoire. We believe that, in order to minimize the chances of a problet generating the same problem more than once for a student during a session, the problet must generate the problems dynamically. In addition:

- **Solving Problems**: The problet must be able to solve the problems it generates. This is required since the problet generates problems dynamically, and is expected to evaluate the answers entered by the user and provide feedback. In other words, the problet must be equipped with a working model of the problem domain.
- **Feedback Generation**: The problet must be able to explain where and why the user's response is incorrect. This level of detail in feedback is necessary to help the users learn from their mistakes.
- **Calibrating the Hardness of Problems**: The problet should be able to cater to the needs of the user. This would mean altering the hardness of the problems to suit the requirements of the user. If the user is experienced then it would not interest the user if the problet were to provide trivial problems. The problet must be capable of calibrating the hardness of the problems based on the needs of the user.

- **Visualization Techniques**: The problet must provide visualization techniques that assist the user in solving the problems. These visualization techniques may be provided to address three different needs:
  - **Transparency of Technology:** The technology behind the problet must be transparent to the user, so that the user can concentrate on solving problems and not be hindered by having to do so on-line. E.g., if the answer to a question involves drawing a diagram, the user should be provided adequate facilities to draw such diagrams online, and should not instead be required to express the same diagrams in words.
  - **Tools at Paper and Pencil Level**: The problet must provide the user with at least the same working environment for solving problems as the user would have when solving problems using paper and pencil. In other words the on-line problem-solving environment must not be inferior to the traditional problem-solving environment. E.g., if the user would be able to view the code and its architecture at the same time on two different pieces of paper, two separate windows with that information must be made available to the user in the problet.
  - **Enhanced Online Environment**: The problet may also provide problem-solving tools that are typically not available off-line, and thereby enhance the online problem-solving environment. E.g., it is easy to provide calculators, text highlighting, color-coding and other such problem-solving aids on-line. Searching and sorting are two techniques normally not available off-line, which can be easily provided on-line.

- **Free and Across Platforms**: The following are *desirable* characteristics of problets in terms of their accessibility and use:

  - **Platform-independent**: The problet must be available across all platforms, so that the students can access it from either the school or home computers. In order to avoid the overhead of porting a problet from one platform to another, the technology used to implement problets must preferably be platform-independent.

- **Inexpensive or Free**: Using a problet should preferably not require the purchase or installation of expensive hardware and/or software, so that it is accessible to students across the economic spectrum. Installation of additional hardware/software may not only be expensive but also potentially stressful to students and may discourage students from using the problets.
- **Distributed Over the Internet**: If the problet can be made available over the Internet, it will be available around the clock, from anywhere: the school, home or an Internet Service Provider. Students do not have to make face-to-face contact with the instructor in order to be able to access the problet. This is especially useful in distance learning courses.

Java meets all these requirements: it is platform-independent, freely available for academic use, runs on the Internet, and requires only a Java-enabled browser to run. Therefore, we have used Java to implement our problets. As added bonuses, Java is safe, the applet environment is secure, applets may be loaded and removed at will and there is no need to install or remove them.

5. Components of a Problet

The components of a typical problet are:

- A problem **template**, which is the blueprint of the types of problems generated by the problet. The template in turn consists of the following components:
  - The **background information** necessary to solve the problem;
  - The ***stem*** of the problem, which states the problem;
  - The **response options** provided to the user;
  - The format of the **feedback** to be provided to the user, which includes the feedback to accompany correct as well as incorrect responses.

  A problet generates problems based on a template, by randomizing the parameters of the template. Heuristics are used during problem generation not only to ensure that the problems generated by the problet are not trivial, but also to calibrate the hardness of the generated problems.

  We use the following definition of hardness when generating problems: "*The more the plausible response options a user has for a problem, the harder the problem*" (Kumar, 1997, Kumar, 1998). E.g., multiple choice problems are harder than true/false problems, and problems with free-form answers are harder than multiple choice problems. Our interest is in generating multiple choice problems, where the more the likely answering options for a question, the harder the problem. The above heuristic definition of hardness of problems has intuitive appeal, is easy to implement, and is therefore convenient for use in problets.

- **Visualization tools**, which support the problem-solving environment of the problet. A well-designed problet may provide several alternative visualizations of a problem (text, graph, etc.), formatting options for these visualizations that help the user focus on the highlights of the problem, and facilities for the user to interact with the visualizations.

Human-factors issues must be taken into account when designing the visualizations. E.g., a problet may display no more than 9 items on the screen at any time, as recommended by Miller's rule of seven plus/minus two (Miller, 1956). (This rule may also be used during problem generation to limit the cognitive load on the user.) Finally, the user interface must be easy to use and intuitively designed.

6. Problet on Static Scope in Pascal

6.1 The Domain

Scope is one of the properties of variables in programming languages. The issue of scope (static or dynamic) arises in all the programming languages. In the Programming Languages course, static scope is typically studied in the context of Pascal programs (Sebesta, 1999). Pascal is considered the ideal language to study static scope because it allows nesting of procedure definitions, and therefore, it is significantly more complicated than most other procedural languages. This problet presents randomly generated Pascal programs and tests the user's understanding of static scope in the programs.

6.2 The Design of the Problet

Background Information: A Pascal program with nested procedure definitions is randomly generated, and displayed in a separate window. Within each procedure, only variable declarations and nested procedures are listed. The syntactic structure of each procedure is:

```
procedure name
  <variable declarations>
  <nested procedure definitions>
 begin {name}
  ...
 end {name}
```

Based on this program, the problet asks the user to identify the scope of a randomly chosen variable declared in a randomly chosen procedure in the program.

Problem Stem: The stem of the problem, along with the response options is displayed in a second window. The stem reads: "Assuming static scoping is used, select all the procedures which lie within the scope of variable **a** declared in the procedure **main**." Note that only the variable name and procedure name are changed from one instance of the problem to the next.

Response Options: The problet lists the names of all the procedures in the program as checkboxes. The user is expected to check all the procedures which (s)he believes are within the scope of the variable, and click on a button called "Check My Answer," when done (See Figure 1).

Feedback: When the user clicks on "Check My Answer" button in the template, the problet solves the problem and compares its solution with the user's response. It then provides feedback by printing the following items in order:

- The procedures selected by the user;
- The procedures in the correct answer;
- The procedures selected by the user which are correct;
- The procedures that the user missed - it explains why these procedures are in the scope of the variable in question;
- The procedures selected by the user that are incorrect - it explains why these procedures are *not* in the scope of the variable in question.

Visualization: The problet provides the following visualization tools to clarify the presented code:

- the user may print the code using a different color for each level of nesting;
- the user may print the code with a bounding box around each procedure (See Figure 2).

The problet is currently posted at http://orion.ramapo.edu/~amruth/java/opl/scope/

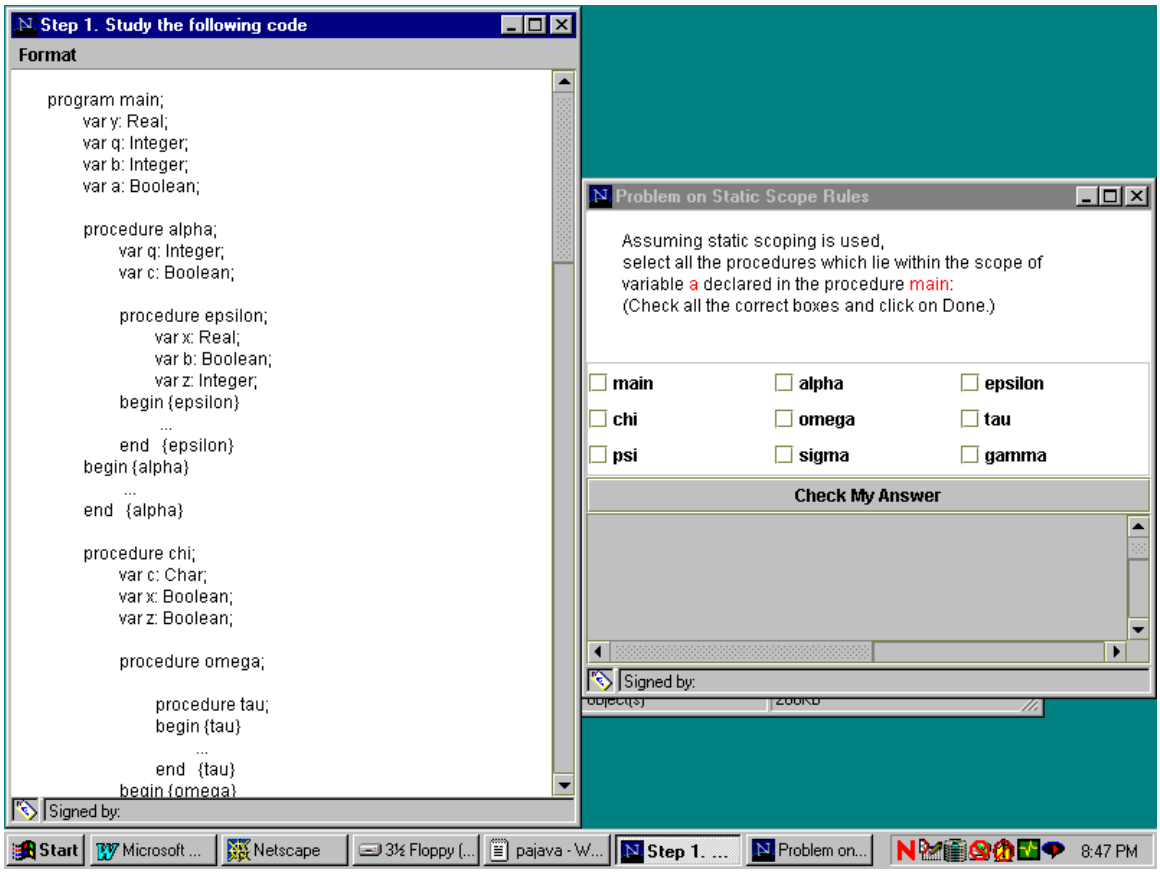

Figure 1: Static Scope Problet: The Generated Code, Problem and Interface

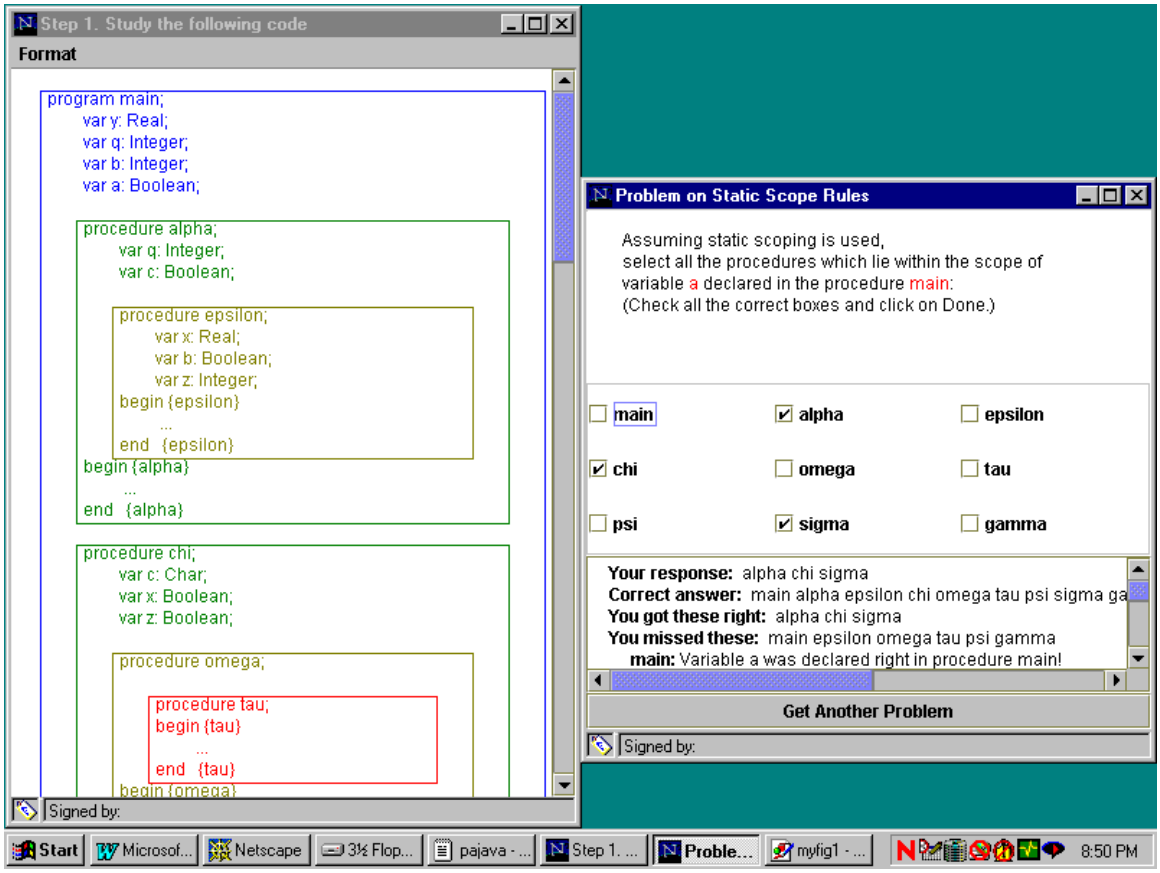

Figure 2. Static Scope Problet: Visualization and Feedback

7. Problet on Nested Selection Statements in C++

7.1 The Domain

In most programming languages, two-way selection statements may be nested to implement multi-way selection. In C++, this is complicated by the fact that C++ does not provide syntax to encapsulate the if-clause and else-clause, which may be either simple statements or compound statements (enclosed in braces). Therefore, students find it hard to trace the logic of nested selection statements in C++. Nested selection statements in C++ also suffer the *dangling else* problem, wherein the association of *else* with *if* is determined not based on syntax but rather on implicit rules. The number of levels of nesting, the presence or absence of indentation, the presence or absence of braces around if-clause and else-clause - all these can complicate a student's analysis of a nested if-else statement. This problet will assist the user in understanding nested selection statements by generating various nested pieces of code and letting the user predict their output.

7.2 The Design

Background Information: The problet generates a nested IF-ELSE statement and displays it in a window. The structure of a C++ selection statement is of the form:

if < condition >
<IF-clause>
else
<ELSE-clause>

The user has the option to have the problet generate problems at three different levels of hardness: basic, intermediate and advanced. The problet starts at the basic level by default. The problet can generate all the following types of nested selection statements: random (no relationship between the conditions of the various statements), classification (relationship exists between conditions of the various statements), chained statements and dangling else statements.

Problem Stem: The stem of the problem, along with the response options is displayed in a second window. The stem reads *"What is printed by the code for the following values of the variables:"* Note that only the values of the variables are changed from one problem instance to the next (See Figure 3).

Response Options: The problet lists the names of all the possible outputs as radio buttons. The user is expected to select the output that the user believes is correct. After that the user can click on the button called "Check My Answer" to obtain feedback.

Feedback: When the user clicks on "Check My Answer" button on the template, the problet verifies the user's response and states whether the user's response is correct or not. If the user's response is incorrect, it lists the correct answer, and lists the reason why the user's response is incorrect.

`Visualization:` The problet provides the following visualization tools to clarify the presented code:

- the user may print the code using a different color for each level of nesting;
- the user may indent the code for easy readability. This feature is however disabled when *dangling else* problems are generated, as otherwise, the indentation would give away the answers to the problems.
- The user may print the code with or without braces around each if-clause and else-clause.

Menus have been kept at a single level so as to keep the user interface simple.

The problet also presents the fork diagram (Kumar, 1996) for the generated code in a separate window (See Figure 4). A Fork Diagram is a graphical representation of one and two way selection statements. Fork diagrams may be used to analyze a given if-else statement and answer questions based on it. This problet is posted at: http://orion.ramapo.edu/~amruth/java/opl/nestedif

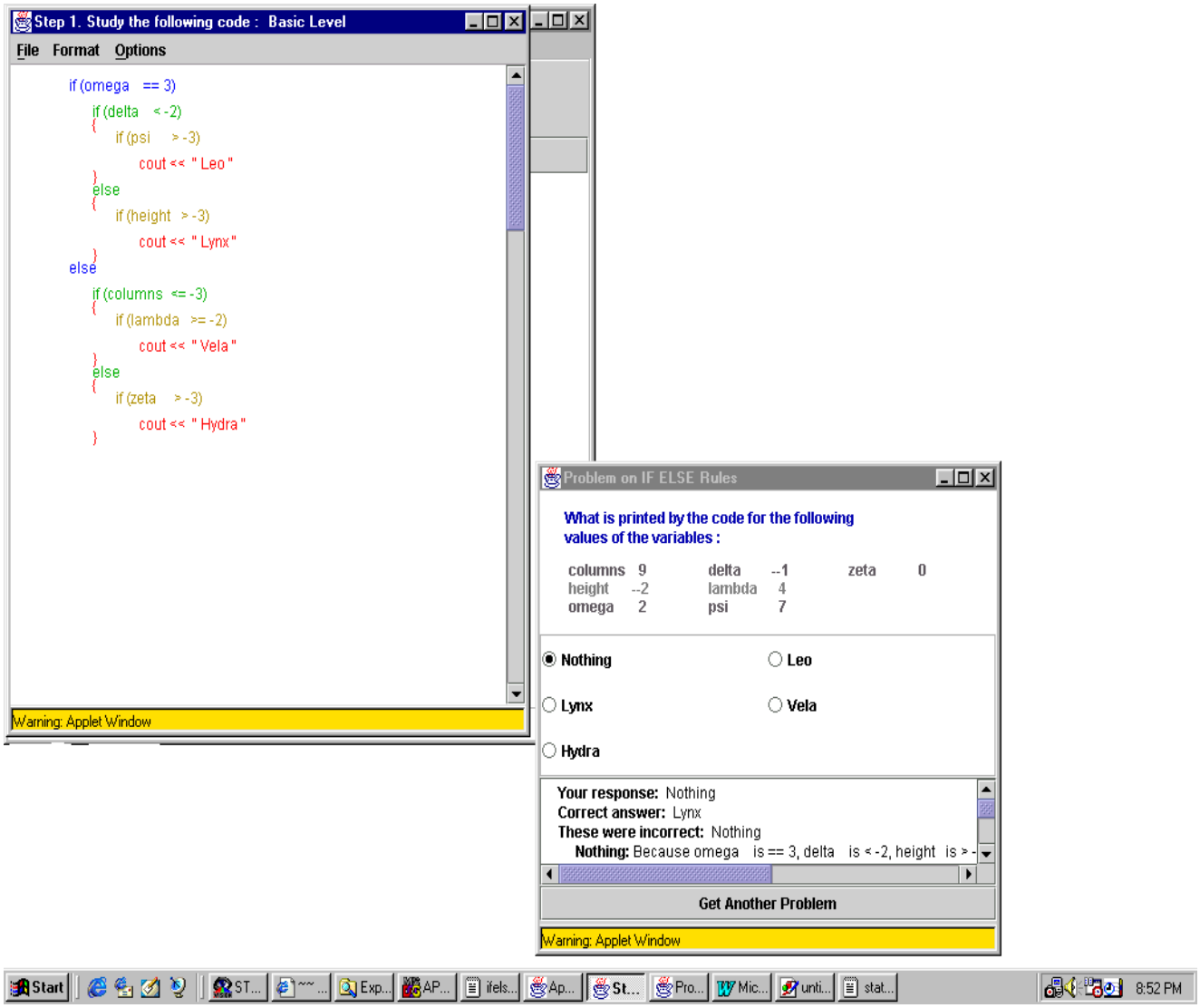

Figure 3: Nested Selection Problet: The Generated Code, Problem, and Interface

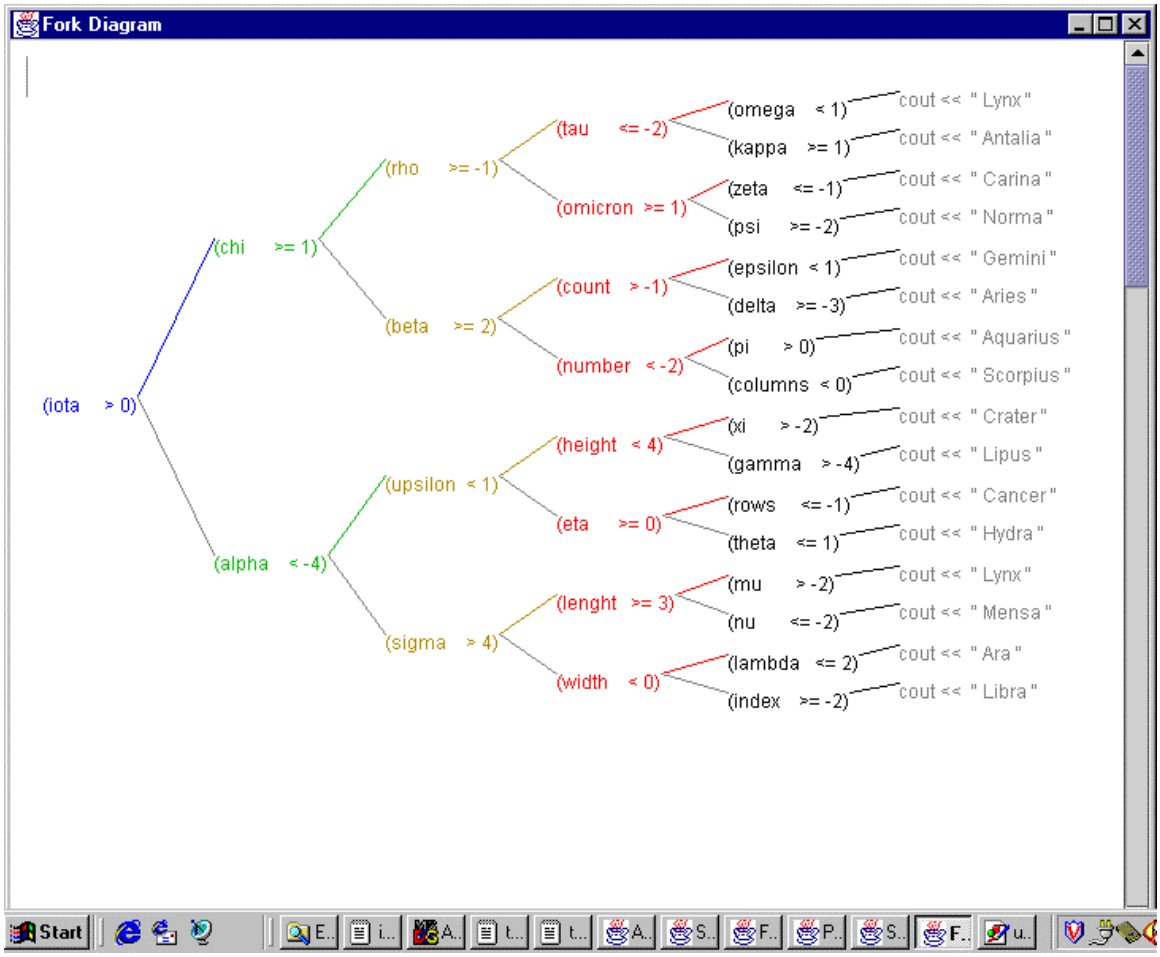

Figure 4: Fork Diagram for a nested selection statement in C++

Future Work

We use Model View Controller (MVC) pattern to improve the reusability and modifiability of our implementations. This pattern also enables us to keep the same set of classes for viewing different models such as the ones for static scope, nested selection statements or some other Computer Science problem. Currently, we are implementing each problet independently, although we have been reusing large amounts of code from one problet to the next. We hope to collect a library of classes from the implementations of the problets and make them available for the development of future problets.

The problets we described in this paper are functional - they generate and solve problems. Future work includes incorporating mechanisms for logging usage by user name. Instructors may use such logs to monitor the assignments they hand out to be carried out on the problets. We also plan to design and implement problets for other topics in Computer Science, and deploy them in our courses.

References

[1] Brusilovsky, V. Task sequencing in an intelligent learning environment for calculus, In proceedings of the Seventh International PEG Conference, Edinburgh, Scotland (July 1993), 57-62.

[2] Buschmann, F,, Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., Pattern Oriented Software Architecture – A system of Patterns, John Wiley & Sons, Ltd. (1996).

[3] Farnworth, C. C., Using computer simulations in problem-based learning. In proceedings of Thirty Fifth ADCIS conference, Omni Press, Nashville, TN, (1994), 137-140.

[4] Koffman, E.B. and Perry J..M. A model for Generative CAI and Concept Selection, International Journal for Man Machine Studies, 8, 1976, 215-234.

[5] Koffman, E.B. and Blount, S. E. Artificial Intelligence and automatic programming in CAL, Artificial Intelligence, 6, (1975), 215-234.

[6] Kumar A.N., Generating Challenging problems in Interactive Tutoring Systems : A Case Study of Storage Placement Algorithms., In Proceedings of the Eighth International PEG Conference, Sozopol, Bulgaria, (May 1997), 128-134.

[7] Kumar A.N., Problem Generation : Evaluation of two domains in Operating Systems, In Proceedings of the International FLAIRS Conference (FLAIRS '98), (Special Track on Intelligent Tutoring Systems), Sanibel Island, FL (May 1998), 178-181.

[8] Kumar, A.N., Fork Diagrams for Teaching Selection in C.S. I, Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education, Philadelphia, PA (February 1996), 348-352

[9] Miller, G.A., The Magical Number Seven Plus or Minus Two: Some limits on our Capacity for Processing Information, The Psychological Review, 63, (1956), 81-97.

[10] Sebesta, R.W., Concept of Programming Languages 4th Ed. Addison Wesley Longman Inc. 1999.

[11] Vernon D.T. Attitudes and opinions of faculty tutors about problem based learning, Academic Medicine, 70(3), 1995, 216-233.

[12] Wenger, E., Artificial Intelligence and Tutoring Systems, Computational and Cognitive Approaches to the Communication of Knowledge, Morgan Kaufman Publishers Inc. 1987.