

**Final Summary Report: Learning Data Structures
INFSCI 2925: Independent Study
University of Pittsburgh
Advisors: Drs. Brusilovsky and Flynn**

**Jeff Kaminski
7 January 2007**

This final report summarizes the work performed for INFSCI 2925, an independent study that represents a continuation of a final project from INFSCI 2470, "Interactive System Design," from the spring 2006 semester. I worked on the independent study through the summer and fall semesters of 2006. My instructor for INFSCI 2470 was Dr. Brusilovsky, and the final project for that class was designed and completed for Dr. Flynn. Both professors graciously agreed to act as advisors for my follow-on independent study.

This summary report includes the following sections:

- **1. Project Overview** provides a high-level description of the project, including the initial goals and deliverables of the independent study.
- **2. Feature Descriptions** describes the interface and its features in great detail. This section represents the bulk of the summary.
- **3. Results of User Testing** details the results of the user testing conducted with members of Dr. Flynn's graduate and undergraduate data structures courses late in the fall semester. While only three participants responded, the data is still worth summarizing.
- **4. Conclusions and Future Directions** presents a very brief summary of the project and discusses some future goals for a potential follow-on independent study.
- **A. User Questionnaire** provides an exact copy of the Microsoft Word-based questionnaire made available to test subjects. Note that all subjects who responded elected to complete the equivalent Web-based questionnaire.

1. Project Overview

Because the applet created for INFSCI 2470 was the genesis of the independent study project, I briefly present that version of the project before describing its evolution for the independent study. Figure 1 shows the high-level interface from the original project. As shown, this version of the project supported only the binary tree data structure in the C programming language. As the remainder of this section and Section 2, “Feature Descriptions,” discuss, the original project had much less functionality than the version created for the independent study.

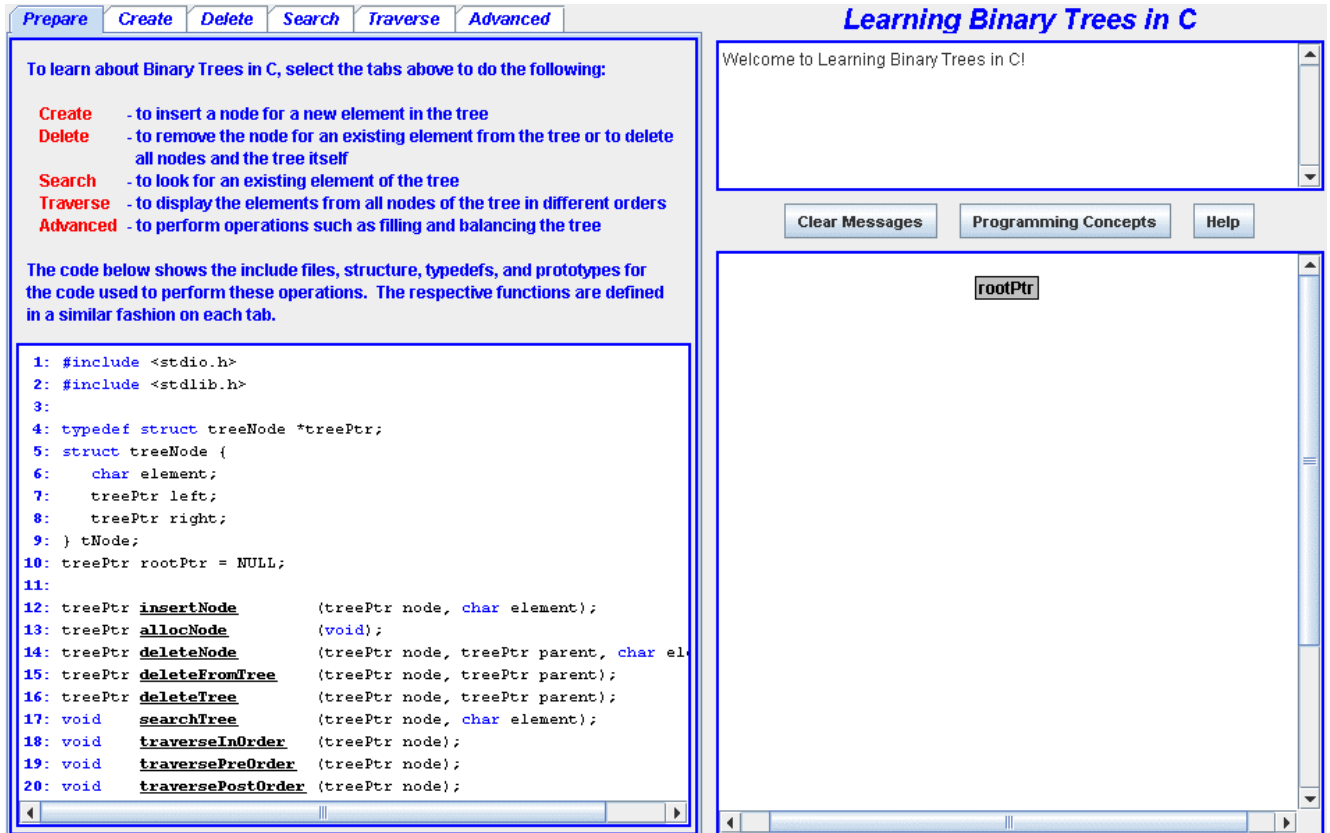


Figure 1: Final Project from INFSCI 2470

Figure 2 shows the high-level interface from the binary tree version of the applet for the C programming language created for the independent study. (Though not shown here, the B-tree version of the interface is very similar, differing primarily in the content of the code pane and the labels of a few of the tabs.) Many differences between the versions of the applet for INFSCI 2470 and for the independent study are immediately obvious. For example, many more buttons providing much richer functionality are available in the latter, and the graphical display of the simple empty tree includes a null pointer to indicate clearly that the tree is empty. A great many differences of greater magnitude are not visible in these simple screen captures; they are described in Section 2.

The target audience for all versions of the applets includes individuals who are learning or interested in refreshing their knowledge of slightly more advanced data structures. The applets assume a basic understanding of programming languages and some familiarity with basic data structures. Beyond that, they are intended to help the user learn and understand the data structures they present with little or no prior knowledge of the structures.

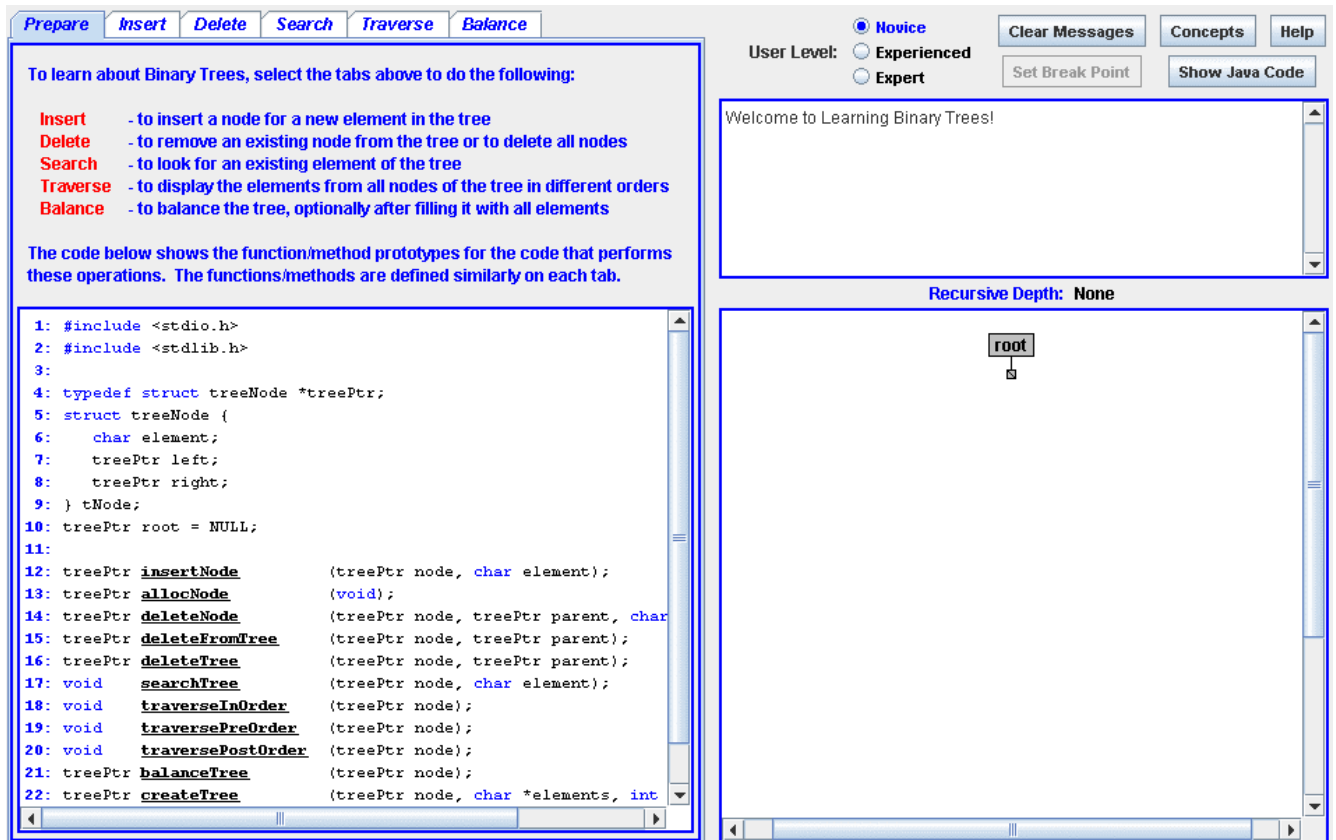


Figure 2: Binary Tree Applet from INFSCI 2925

Both versions of the applet are infinitely adaptable in terms of the operations the user can perform on the tree structures. The user is free to perform and trace any operation on any element, and to select and execute operations in any order. None of the operations are pre-programmed, so the user drives execution of the actual code for the tree, meaning he or she can manipulate the data structure in whatever means is desired.

The representation of the structure is both physical and virtual. The applet code is executing on an actual physical structure, performing and displaying the results of each step of the execution to the user. But the underlying structure is always a Java-based tree that is typically manipulated in additional ways not visible to the user to facilitate the virtual display presented via the interface. So conceptually, the user is seeing exactly what the code is doing to a real structure, but that structure is virtualized for more meaningful presentation to the user.

The independent study greatly enhanced the adaptive, personalizable capabilities of the project by introducing selectable user levels. Three levels, Novice, Experienced, and Expert, affect the amount and scope of messaging displayed as code executes. The user levels also impact the rate of execution for All mode. While more could be done with this feature, the initial function does provide a level of personalization and customization that tailors the user experience to the user's level of understanding.

1.1 Project Deliverables

I committed to provide the following high-level deliverables for the project:

- Functional code that achieves the project goals of the proposal. All code will be fully tested and delivered in a fashion that lends itself to installation on a Web server.
- A project report that summarizes the results of any initial investigation, the design goals and requirements, the resulting implementation, and the results of user testing. The document will be delivered in Microsoft Word format, probably with additional attachments.

1.2 Project Goals

Table 1 lists and describes the original project goals in terms of the final project for INFSCI 2470 and the status of each as attained for the independent study. Some of the features weren't attempted, primarily because other features not listed here were deemed more valuable. Introducing break points, adding the ability to cancel an operation, and developing a more interesting graphical representation of the more complex B-tree structure are examples of additional features that, while not part of the original goals, proved more valuable to the success of the resulting applets.

	Project Goals	Status
1	Functional Enhancements	
1.1	Add support for B-trees. To complement the existing binary tree work, the project will be extended to demonstrate B-trees as well. The B-tree is a more complex structure, but it is also extremely powerful. Adapting the existing interface to include B-trees will markedly increase the utility of the applet.	Completed in its entirety.
1.2	Add support for the Java programming language. In addition to providing C source code for both the binary tree and B-tree, the applet will present the same level of functionality via Java source code. The end result will be support for both structures in both languages.	Completed in its entirety.
1.3	Improve the implementation of the project. At the implementation level, introduce a more modular architecture to facilitate incorporation of additional structures and languages. Currently, the binary tree is wedded a bit too closely to the interface. Redesigning certain aspects of the project can move the overall effort closer to the model-view-controller paradigm.	While innumerable aspects of the source code were redesigned and improved, I was not able to modify the code to the point where new structures could be plugged in easily. New languages are much easier to introduce, however.
1.4	Enable the advanced operations. The applet was intended to include a demonstration and discussion of the advantages of a well-balanced tree versus an unbalanced tree. The "advanced" operations not implemented for the final project include balancing a tree and populating it with a full set of nodes for all available elements. In the latter case, adding nodes in either a random or ordered fashion creates a well-balanced or sub-optimal tree. Enabling the advanced operations lends itself well to a discussion of Big O notation in terms of search times through the tree (Order (log N) versus Order (N)). This functionality will be provided for both binary trees and B-trees.	All advanced operations were added, though the nature of the B-Tree doesn't lend itself well to discussions of ordered versus random filling, and the tree is generally balanced by default, at least as shown in the applet. For search operations, messages also present the actual and maximum number of nodes examined.
2	Interactive Adaptive Enhancements	
2.1	Ascertain the user's level of knowledge. To encourage and support a more adaptive interface, ascertain the user's level of knowledge and use that to customize aspects of the interface. For example, users could be classified into three tiers: beginner / novice, average / intermediate, and experienced / knowledgeable. These classifications could then be used, for example, to adapt the explanatory messages provided by the interface to the user's level of sophistication. For this goal, I would initially envision allowing the user to indicate his or her level of knowledge when the application starts, rather than trying somehow to test the user since I'm not convinced I can develop a truly effective means of evaluation. User levels could be captured by the applet on a per-session basis, since they're likely to be indicated via a very simple set of mouse clicks. The alternative would be to enable persistent storage of user profiles on a backend server. However, the added work of developing a backend servlet might reduce the time available to develop the interface itself, adversely impacting the scope of the interface, which is where I'd prefer to focus my efforts. That said, this is something that can be discussed further.	As mentioned previously, three users levels were introduced, and the user can indicate his or her level of understanding from the initial selection page for the project or dynamically at any time while using the applets. User levels apply only on a per-session basis; they're not stored persistently.
2.2	Provide different, customizable levels of explanatory messages. Based on the user's level of knowledge, present different levels of explanatory messages. Currently, the application presents only terse explanations. This feature would provide different levels of explanation	Completed in its entirety.

	tailored to different types of users, with expansive descriptions shown to beginners, less detailed messages shown to average users, and only brief messages presented to expert users. On a similar note, store the explanations in an external repository from which they can be modified by the developer or instructor to suit his or her needs.	
2.3	Let the user predict the results of operations. Allow the user to predict the results of operations. For example, when adding an element to a structure, let the user predict the parent node for that element and the position of the new child. When deleting a node, let the user indicate the results of the operation (e.g., which node will replace the one to be deleted). This may be more practical for binary trees than for B-trees, but it would be valuable in both contexts. One possible means of achieving this goal would be to make the nodes of the drawing selectable, which could also offer additional benefits.	Not attempted.
3	Interface Enhancements	
3.1	Improve the drawing of a structure. Primarily, the drawing of a structure can be improved by better reflecting the actual pointers between the nodes. Specifically, showing the null pointers associated with any node will make the graphical flow of operations clearer; as it stands today, exploring a null pointer during execution of an operation simply means that no node is selected, which can be somewhat disconcerting. Additionally, the flow of traversal over a structure can be demonstrated more effectively. This could be accomplished by highlighting the links between nodes of the tree and/or providing directional indicators (e.g., up and down arrows) of the direction of flow of traversal. I can also explore the use of different colors to indicate the flow.	Null pointers were added in all cases where they appear in the structure, greatly improving the user's understanding of operations as they traverse the nodes of the tree. Highlighting is used for nodes, elements (in the case of B-trees, where each node can have multiple elements), and null pointers. Links are not highlighted, since that didn't seem to be warranted once other highlighting was in place.
3.2	Present recursive depth more effectively. Currently, the depth of recursion of any operation is presented in the form of a message. This approach tends to fill the message area with comments of this nature. Given how frequently such messages are generated and the importance of understanding the recursive nature of the operations, it would be more effective to indicate the current recursive depth via an alternative means, perhaps in a static area of the interface.	Completed in its entirety via the addition of a static display between the message and drawing areas of the interface. For B-trees, the applet reports the complete call stack, not just the depth of recursion.
3.3	Provide more flexibility to the user. The areas of the interface (code area, message area, drawing area) are currently static; the user cannot resize them. Allowing the user to modify the dimensions of any area would let the user focus on the specific aspect of the applet in which he or she is most interested.	Not attempted.
3.4	Incorporate some cosmetic enhancements. This includes finding different names for the Next Step, All Steps, and No Steps buttons; providing customization of font sizes and colors; and supporting different rates of execution for stepping through the execution of an operation. These are largely cosmetic and don't significantly alter the nature of the application, so they're of the lowest possible priority.	Many cosmetic changes were introduced, resulting in a clearer display and a more attractive interface. Of the items mentioned, only customizable font sizes and colors were not added.

Table 1: Summary of Project Goals

In addition to these changes to the applets, I'd committed to user testing and evaluation, as follows:

Once the applet is functional, I will conduct user studies to evaluate the interface. The studies would be conducted with a variety of users, focusing on subjects from the actual target user community (i.e., university students at the undergraduate and/or graduate levels). The results of the studies will be collected and analyzed to determine the effectiveness of the application and to identify potential enhancements. Formal surveys and data collection techniques will be used.

See Section 3, "Results of User Testing," for information about the testing and the results it yielded.

2. Feature Descriptions

This section provides high-level information about the features of the final applets. Table 2 lists the most significant features of the applets. The Feature column briefly describes each feature, and the Status column indicates whether the feature existed from INFSCI 2470 or is new for the independent study. Note that all features are new for the B-tree data structure, regardless of whether they existed previously for the binary tree. The Section column identifies the section of this report in which the feature is described in more detail. The final column, Notes, provides brief comments about some of the features.

As commented in the previous section, virtually all code from the first version of the project was completely rewritten for the independent study. A number of defects were found and fixed, but more importantly, the code was reorganized and heavily rewritten to support the new features in an elegant fashion. It would not be incorrect to consider all of the code as new for the independent study.

Feature	Status	Section	Notes
Support for binary tree data structure	Existing	2.1	Existing but modified heavily throughout source code
Support for B-tree data structure	New		All B-tree functionality is new, regardless of whether the functionality is labeled as existing for the binary tree
Definitions and prototypes	Existing	2.2	
Insert node/element	Existing		
Delete node/element	Existing		
Delete entire tree	Existing		
Search for element	Existing		
Traverse tree in-, pre-, and post-order	Existing		B-tree support for in-order traversal only
Fill tree with random or ordered data	New		B-tree support for fill with ordered data only
Balance tree	New		Binary tree support only
Code tracing	Existing	2.3	
Descriptive user messages and clearing	Existing	2.4	
Graphical tree with highlighting	Existing	2.5	
Inclusion of null pointers in graphical tree	New		
Description of recursive depth / call stack	Existing	2.6	Recursive depth shown for binary tree, entire call stack shown for B-tree
Support for C-language code	Existing	2.7	
Support for Java-language code	New		
Dynamic runtime selection of language	New		
Next (single-step) execution mode	Existing	2.8	
All (multiple-step) execution mode	Existing		
Silent (instantaneous) execution mode	Existing		
Undo step for execution	New	2.9	Not yet implemented for either structure
Cancel execution	New		
Support for different user levels	New	2.10	
Dynamic runtime selection of user level	New		
Support for setting / clearing break points	New	2.11	
Conceptual documentation linked from code	Existing	2.12	Modified for new features
Help documentation	Existing		Modified for new features
Introductory JavaScript selection page	New	2.13	

Table 2: Summary of Project Features

2.1 Data Structures

As mentioned previously, the project supports both binary tree and B-tree data structures. The two data structures are supported via two independent applets, with separate source code for each. Almost half of the code for each applet is the same or very similar to that of its counterpart. The remainder is very different. A possible future enhancement would be to merge common code from the two applets to minimize redundancy, but that was not critical for this independent study.

2.2 Operations on Data Structures

Each applet supports the primary operations one would expect for its data structure. The following subsections describe the supported operations at a high level, discussing them in terms of the tabs into which they're organized. Many of the tabs are very similar between the binary tree and B-tree applets, but the differences are appreciable enough in many cases to warrant showing both applets.

2.2.1 Prepare Tab

The Prepare tab for each applet briefly describes the operations and functionality of the applet. Figure 3 shows the descriptive text for the Prepare tab of the binary tree applet. The Prepare Tab of the B-tree applet is very similar.

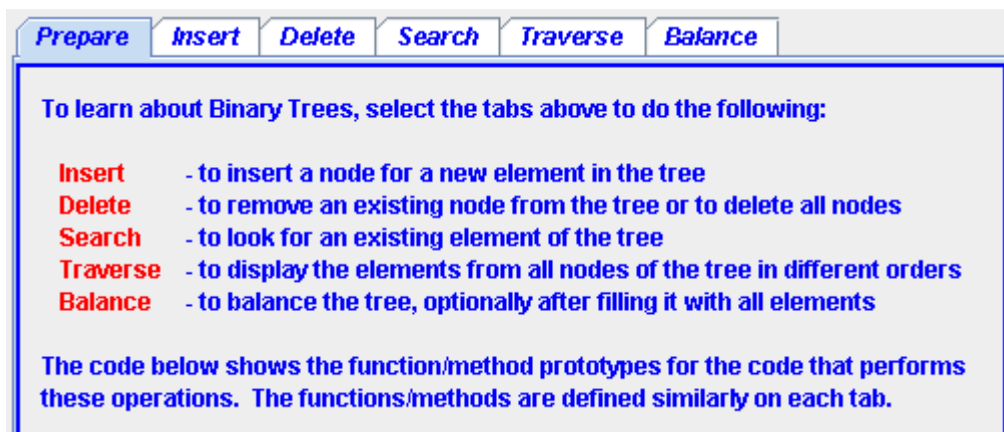


Figure 3: Binary Tree Prepare Tab

For each applet, the introductory text is followed by include statements; definitions of constants, variables, and structures; and the prototypes for the functions and methods associated with the data structure in the current programming language. The Prepare tab is always the first tab shown to the user when the applet is started. See Figure 2 for an example of the Prepare tab as it appears in the complete applet for a binary tree.

2.2.2 Insert Tab

The Insert tab allows the user to insert a new element into the data structure. The user first selects the element to insert. He or she then uses the Next, All, or Silent button to execute and show the operation. (See Section 2.8, "Execution Modes," for more information about executing an operation.) The user can attempt to insert an element already present in the tree to see how the code reacts to such a situation.

Figure 4 and Figure 5 show the Insert tabs of the two applets. The only difference between the two applets concerns the number of elements available for insertion in the tree. Due to the nature of the structures, the binary tree supports a maximum of 16 elements, letters A through P, while the B-tree supports insertion of all 26 alphabetic characters.

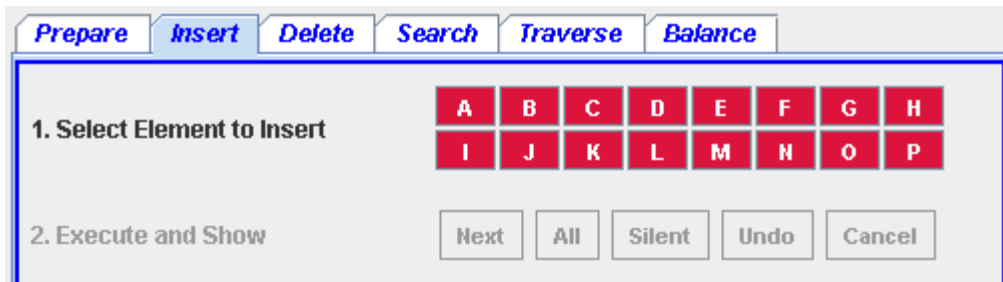


Figure 4: Binary Tree Insert Tab



Figure 5: B-Tree Insert Tab

2.2.3 Delete Tab

The Delete tab lets the user delete an existing element from the tree by selecting first an element and then an execution mode. The user can attempt deletion of an element not present in the tree to see how the code reacts. Figure 6 and Figure 7 show the Delete tabs of each applet. In addition to the difference in supported ranges of characters, the binary tree also includes an All button to let the user delete the entire tree. For the B-tree, this operation is provided on the Tree Operations tab; see Section 2.2.5, "Remaining Tabs," for more information.

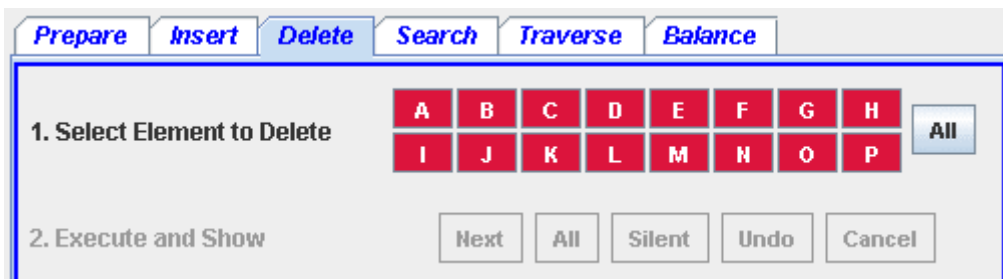


Figure 6: Binary Tree Delete Tab



Figure 7: B-Tree Delete Tab

2.2.4 Search Tab

The Search tab enables the user to search for an element in the tree, again by selecting first an element and then an execution mode. The user can attempt to search for an element not present in the tree to

see how the code responds. Figure 8 shows the Search tab for the binary tree applet. The Search tab of the B-tree applet is identical with the exception of the additional characters it supports.

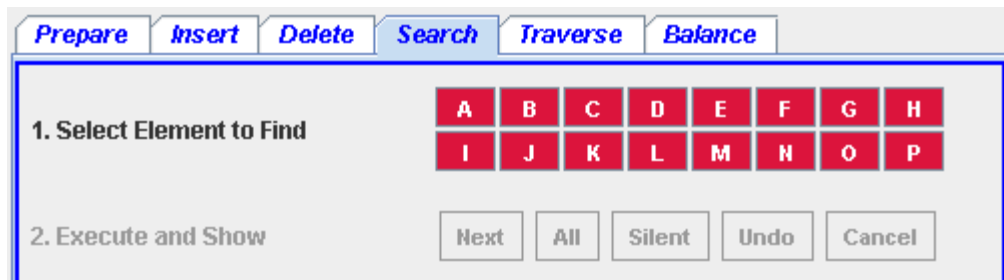


Figure 8: Binary Tree Search Tab

The applets also present the number of nodes examined and the maximum number of nodes that could be examined for a search operation. This information lends itself well to consideration of the Big O qualities of the respective structures, along with the qualities of well-formed (balanced) versus ill-formed trees. For example, a successful search of a binary tree for the element J could result in a message such as “**Element J found in tree (after 6 out of a possible 8 calls)**”.

2.2.5 Remaining Tabs

The remaining tabs of the applets differ to a larger extent. Figure 9 shows the Traverse tab of the binary tree applet, which allows the user to traverse all elements of the tree in in-order, pre-order, or post-order fashion.



Figure 9: Binary Tree Traverse Tab

Figure 10 shows the Balance tab of the binary tree applet. This tab allows the user to fill the tree with all remaining elements (those not already present in the tree) in a random or ordered manner. The difference is significant for a binary tree, since the nature of the structure potentially results in a very different tree for each type of fill operation. The Balance Tree button lets the user create a balanced tree from all of the elements currently present.



Figure 10: Binary Tree Balance Tab

Figure 11 shows the fifth and final tab of the B-tree applet, Tree Operations. This tab includes buttons for traversing all elements of the current structure, deleting all elements from the current structure (and thus the tree itself, a feature available from the All button of the Delete tab for the binary tree applet), and filling

the tree with all remaining elements. For a B-tree, only in-order traversal is supported, since pre- and post-traversal make less sense for such a structure. Also, only an ordered fill of the tree is supported, since the tree remains relatively balanced regardless of the order in which elements are added.



Figure 11: B-Tree Tree Operations Tab

The various fill operations are provided to allow for more meaningful experimentation with the structures. For example, the user can populate a tree with all elements and then conduct more illustrative deletion, search, and traversal operations. Because they are made available as supporting operations, the fill operations are not traced and no code is provided to illustrate them.

2.3 Code Tracing

Code tracing is the first of the four areas of the interface that help illustrate an operation for a user. As an operation is performed, each line of code in the code pane is highlighted as it executes. The user can follow execution of the code to learn the implementation of the structure. Figure 12 shows an example of code tracing from the Delete tab of the B-tree applet. In this case, the current line of code spans two lines, and both are highlighted.

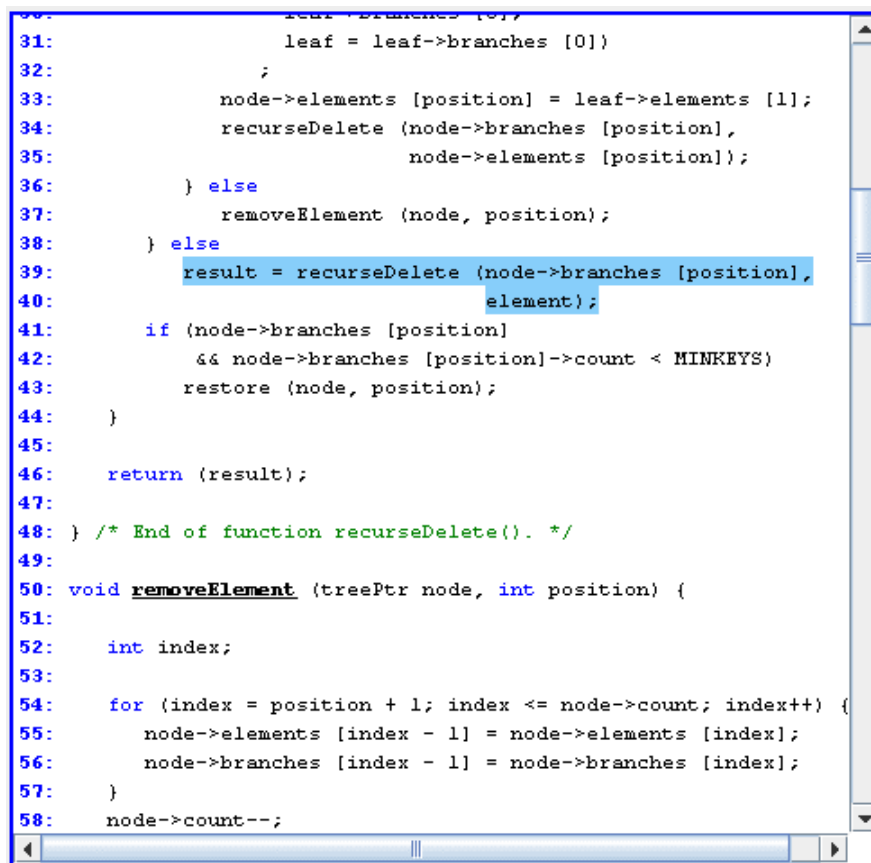
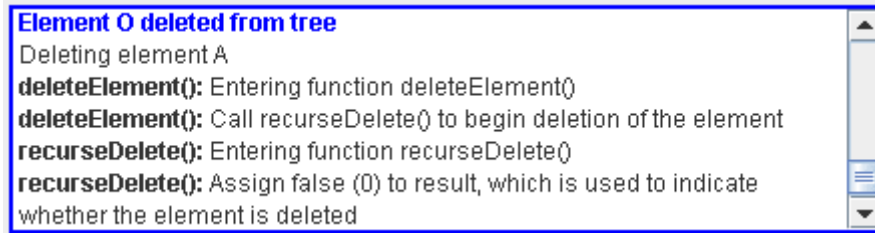


Figure 12: Code Tracing from B-Tree Deletion

2.4 User Messages

User messages are the second means of illustrating an operation. As lines of code are executed, appropriate messages describing the code are displayed in the message area of the interface. Figure 13 shows messages associated with a deletion operation from the B-tree applet. The scope of the messages displayed is controlled by the user level; see Section 2.10 for more information. These messages were displayed for a Novice-level user.



```
Element O deleted from tree
Deleting element A
deleteElement(): Entering function deleteElement()
deleteElement(): Call recurseDelete() to begin deletion of the element
recurseDelete(): Entering function recurseDelete()
recurseDelete(): Assign false (0) to result, which is used to indicate
whether the element is deleted
```

Figure 13: Message Area from B-Tree Deletion

Messages describing lines of code or selected operations are shown in normal font. Messages corresponding to print statements in the code are shown in bold blue or red, depending on whether an operation succeeds or fails (for example, whether an element is successfully added to the tree or cannot be added because it is already present). Error messages are shown in normal red font. The user can select the Clear Messages button (not shown here) to clear all current messages from the message area.

2.5 Graphical Display

The graphical display of the current data structure is the third significant means of illustrating operations and concepts for the user. As the user builds or in any way modifies a structure, the graphical display is updated to reflect the current structure of the tree. Moreover, as operations are performed, the current node, element, or null pointer referred to by the code is highlighted to help the user follow along in the virtual structure. When the user completes an insertion or search operation, the newly inserted or located element remains highlighted after the operation completes. Figure 14 shows a binary tree structure in a quiescent state, with no operation in progress.

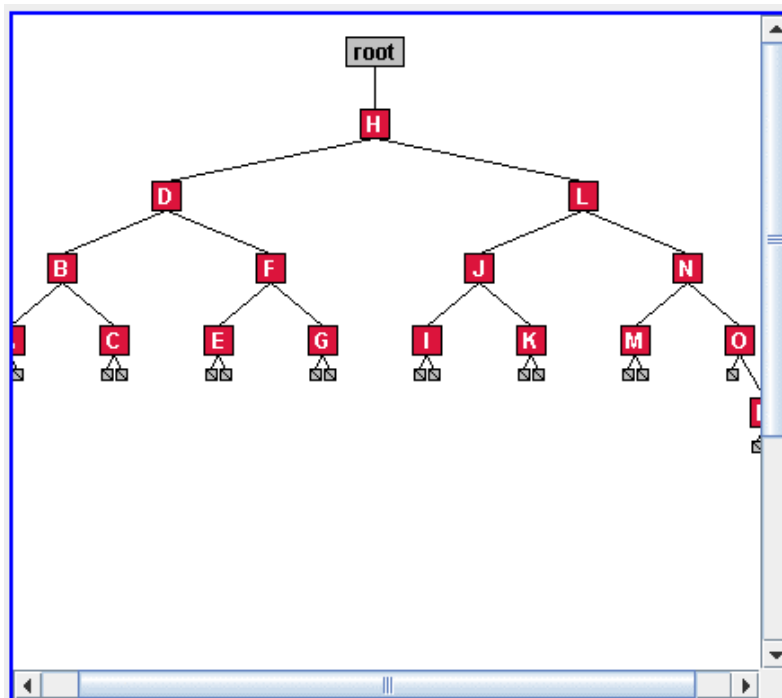


Figure 14: Binary Tree Graphical Display

Figure 15 shows a B-tree structure in mid-operation, a deletion operation in this case. The node containing elements J and K was just repositioned to become the fourth child of the node containing elements C, F, and I. Both the old and new positions of the node are highlighted in blue. In addition, the gray “memory” area at the bottom of the display shows that the right pointer in memory now points to the node that contains L, so that node and its children are shown in the memory area, and the appropriate child node indicating the previous position of the newly relocated node is highlighted as well. The highlighting of nulls supports greater understanding by the user by making connections between steps of an operation and nodes of a tree even clearer.

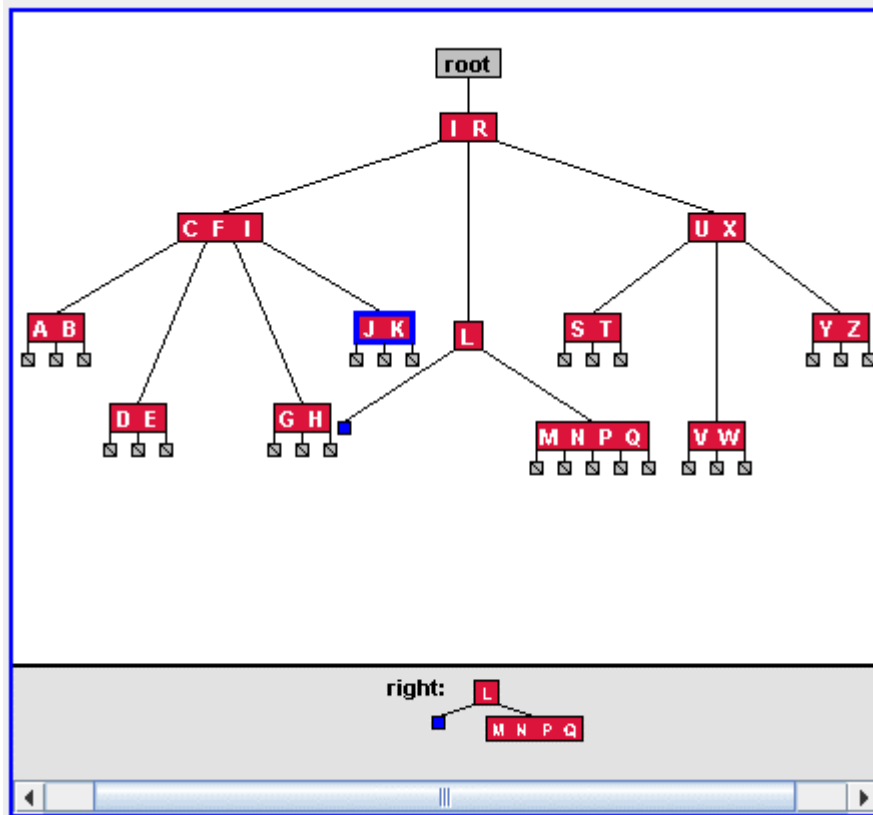


Figure 15: B-Tree Graphical Display

2.6 Recursive Depth and Call Stack

The final aspect of the interface that illustrates an operation concerns the representation of the recursive depth or call stack of the code as it executes. Figure 16 shows the recursive depth of a deletion operation for a binary tree, including the number of times each function or method has been called.

Recursive Depth: deleteNode(): 1, deleteFromTree(): 2

Figure 16: Binary Tree Recursive Depth

Figure 17 shows the call stack from a deletion operation for a B-tree. The entire call stack is shown for the B-tree structure because the code for that structure relies more on individual function and method calls than on recursion. Nonetheless, the number of calls into a function or method is also indicated.

Call Stack: deleteElement(): 1, recurseDelete(): 1, restore(): 1, combineNodes(): 1

Figure 17: B-Tree Call Stack

2.7 Programming Languages

Both applets let the user show and execute code in either the C or Java programming language. Figure 18 shows the code selection button, the label of which changes to indicate the language not currently shown. In other words, the button is labeled Show Java Code while C code is displayed in the code pane; otherwise, it reads Show C Code.

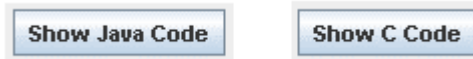


Figure 18: Show Code Buttons

Figure 19 shows the insertion code from the binary tree applet in the C programming language.

```
1: treePtr insertNode (treePtr node, char element) {
2:
3:     if (node == NULL) {
4:         if (node = allocNode ()) {
5:             node->element = element;
6:             node->left = node->right = NULL;
7:             printf ("Element %c inserted in tree\n", element);
8:         } else {
9:             printf ("Cannot add node: insufficient memory\n");
10:            exit (1);
11:        }
12:    } else if (element == node->element)
13:        printf ("Element %c already present in tree\n", element);
14:    else if (element < node->element)
15:        node->left = insertNode (node->left, element);
16:    else
17:        node->right = insertNode (node->right, element);
18:
19:    return (node);
20:
21: } /* End of function insertNode(). */
22:
23: treePtr allocNode (void) {
24:
25:     return ((treePtr) malloc (sizeof (tNode)));
26:
27: } /* End of function allocNode(). */
```

Figure 19: Binary Tree Insertion Code in C

Figure 20 shows the code for the same operation in the Java programming language.

```
1: private TreeNode insertNode (TreeNode node, char element) {
2:
3:     if (node == null) {
4:         node = new TreeNode (element);
5:         System.out.println ("Element " + element + " inserted in t:
6:     } else if (element == node.element)
7:         System.out.println ("Element " + element + " already presen
8:     else if (element < node.element)
9:         node.left = insertNode (node.left, element);
10:    else
11:        node.right = insertNode (node.right, element);
12:
13:    return (node);
14:
15: } // End of method insertNode().
16:
17: private TreeNode (char element) {
18:
19:     this.element = element;
20:     node.left = node.right = null;
21:
22: } // End of method TreeNode().
```

Figure 20: Binary Tree Insertion Code in Java

The user can switch between the two programming languages at any time other than when an operation is currently executing.

2.8 Execution Modes

On all but the Prepare tab, each applet includes Next, All, and Silent buttons to let the user execute a selected operation. Figure 21 shows the three buttons, which are the same for each applet. In the figure, the Next button currently has focus, meaning it can be repeated by pressing the Enter key. The user can press the Tab key to change focus to one of the other enabled buttons.



Figure 21: Execution Mode Buttons

The buttons support the following three execution modes:

- The *Next* button executes one line of code. It highlights the line currently executing in the code pane, displays messages in the message area, and updates the graphical display and recursive depth or call stack accordingly. The user must continue to click on the button to trace through the code one step at a time.
- The *All* button executes and highlights each line of code executed to perform the requested operation, highlighting and updating the interface as described for the Next button. The applet highlights each line of code for approximately one second, though actual execution time varies by user level (see Section 2.10). While an operation is executing, the label of the button changes to Pause, and the user can select it to pause the operation at any time.

- The *Silent* button completes the requested operation instantly, without highlighting any code or otherwise updating the interface. The button is useful, for example, for populating (filling) a structure with nodes and elements before tracing more elaborate operations.

The user can employ any or all of these buttons with any operation other than the fill operations, which are not traced and thus can be executed in Silent mode only. All and Silent modes always execute an operation to completion or until either a break point is encountered (see Section 2.11) or, for All mode, the Pause button is selected. The buttons become active only when an element or operation is selected. However, until one of the buttons is selected to begin an operation, the user is free to select a different element or operation, or to select a different tab of the interface.

2.9 Undo and Cancel

Figure 22 shows the Undo and Cancel buttons that accompany the execution mode buttons described in the previous section. The Undo button is not currently supported; selecting it elicits a message to that extent. The Cancel button lets the user stop a currently executing operation, restoring the structure to the state it had prior to the start of the operation. In effect, selecting Cancel causes the application to act as if the operation was never begun.



Figure 22: Undo and Cancel Buttons

Once an operation begins, selecting the Cancel button is the only way to terminate its execution without letting it run to completion.

2.10 User Levels

User levels provide the most significant aspect of personalization available with the applets. Figure 23 shows the radio buttons used to select a user level; the currently selected level, in this case Novice, is highlighted in blue.



Figure 23: User-Level Selection Radio Buttons

The three supported levels have the following effects:

- The *Novice* level is shown the most verbose output in the message area of the interface. Messages are displayed for virtually all lines of code, and the messages themselves are presented in simple language to make them as clear as possible. When executing in All mode, the applet highlights each line of code for 1.5 seconds.
- The *Experienced* level is shown less verbose messages. Messages are displayed only for decision points (such as iterative or conditional statements) or more complex lines of code. The messages themselves are terser and rely on mathematical symbols to describe relationships and inequalities. When executing in All mode, the applet highlights each line of code for 1.25 seconds.
- The *Expert* level is shown no messages other than those displayed by print statements in the code being executed. When executing in All mode, the applet highlights each line of code for only 1 second.

The user can change user level at any time, and the effects of the new level are immediate.

2.11 Break Points

Break points provide a means of stopping execution of an operation at a selected line of code. When executing in All or Silent mode, the applet stops executing as soon as it reaches a break point. (Break points are ignored in Next mode, and they are not supported on the Prepare tab, where code is not executable.) Break points are useful for stopping execution in the middle of a long operation, avoiding the need to execute each line of code leading up to the line at which the user wishes execution to stop. For a break point to stop execution of an operation, it must occur on a line of code that is executed during the operation.

To set a break point, the user first selects a line of code in the code pane of the interface and then selects either the Set Break Point button or the Set Break Point option from the popup menu available from right-clicking in the code pane. To clear a break point, the user selects either the Clear Break Point button (the label of the Set Break Point button changes to Clear Break Point when a break point is set) or the Clear Break Point option available from the popup menu. Figure 24 shows the two buttons and the popup menu for break point setting and clearing.



Figure 24: Break Point Selection

When a break point is set, the line number of the line of code on which the break point is set is highlighted in red. In Figure 25, a break point has been set on line 8 of the deletion code from the binary tree applet.

```
1: treePtr deleteNode (treePtr node, treePtr parent, char element)
2:
3:     if (node == NULL)
4:         printf ("Element %c not found in tree\n", element);
5:     else if (element == node->element) {
6:         node = deleteFromTree (node, parent);
7:         printf ("Element %c deleted from tree\n", element);
8:     } else if (element < node->element)
9:         node->left = deleteNode (node->left, node, element);
10:    else
11:        node->right = deleteNode (node->right, node, element);
12:
13:    return (node);
14:
15: } /* End of function deleteNode(). */
16:
17: treePtr deleteFromTree (treePtr node, treePtr parent) {
18:
19:     treePtr successor;
20:     int left = 0;
21:
22:     if (node->left == NULL && node->right == NULL) {
23:         free (node);
24:         return (NULL);
25:     } else if (node->left != NULL && node->right == NULL) {
26:         successor = node->left;
27:         free (node);
```

Figure 25: Break Point Set for Binary Tree Deletion

Break points are automatically cleared when the user selects a new tab of the interface or changes the programming language. They can be set or cleared while an operation is running. Finally, the popup menu allows the user to set a new break point without first clearing an existing break point, something that cannot be done from the buttons of the interface.

2.12 Documentation

Both applets provide two types of documentation. The first is conceptual documentation about the code available for execution. In addition to providing an overview of the data structure and an introductory paragraph for each function and method, the conceptual documentation describes each function or method in terms of its lines of code. Figure 26 shows the concepts documentation for the deletion code of the B-tree applet.

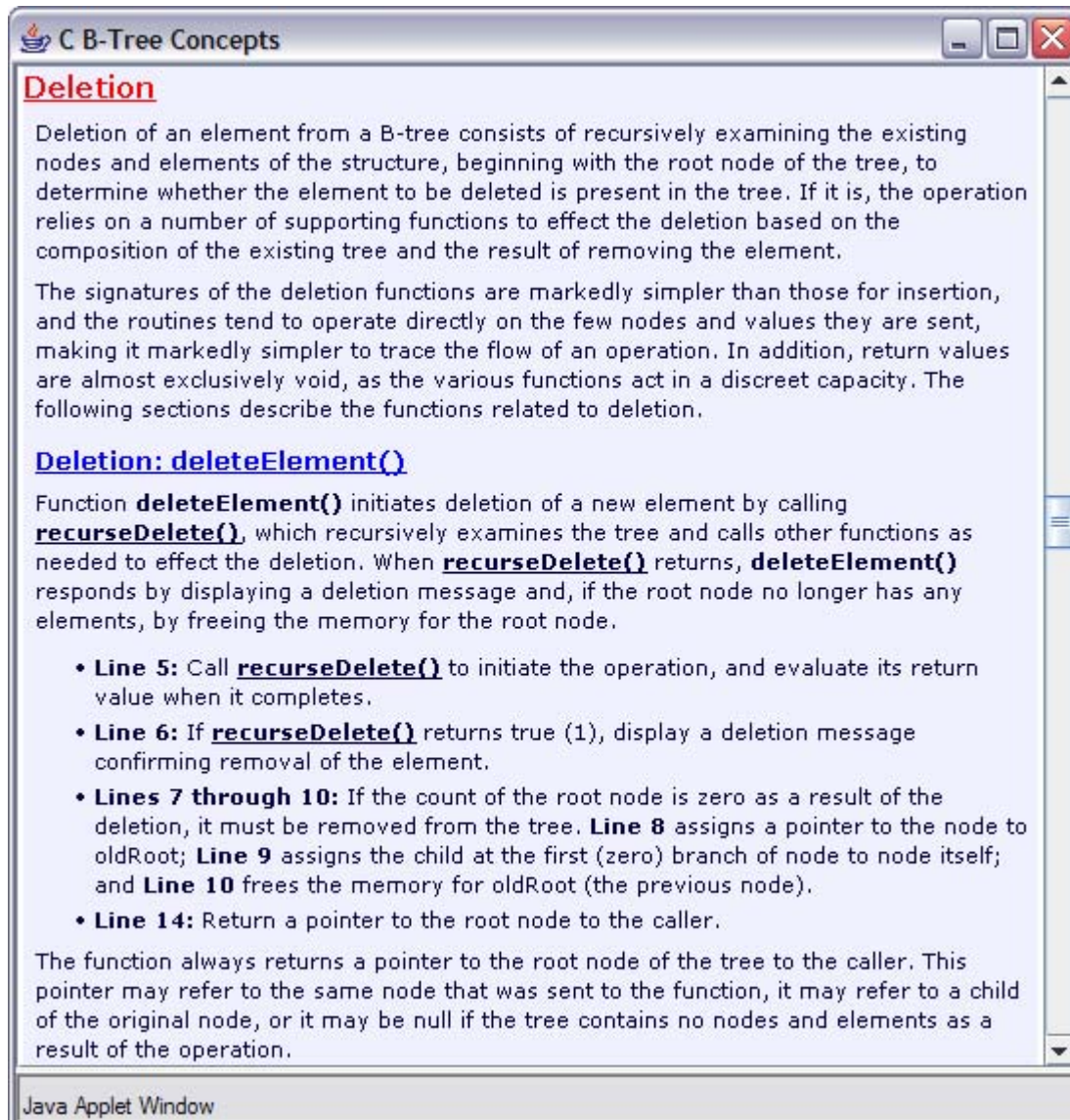


Figure 26: Concepts Documentation from B-Tree Deletion

The second type of documentation is help text for the applet itself. The help describes the primary features and usage of the applet. Figure 27 shows the help text for the binary tree applet.

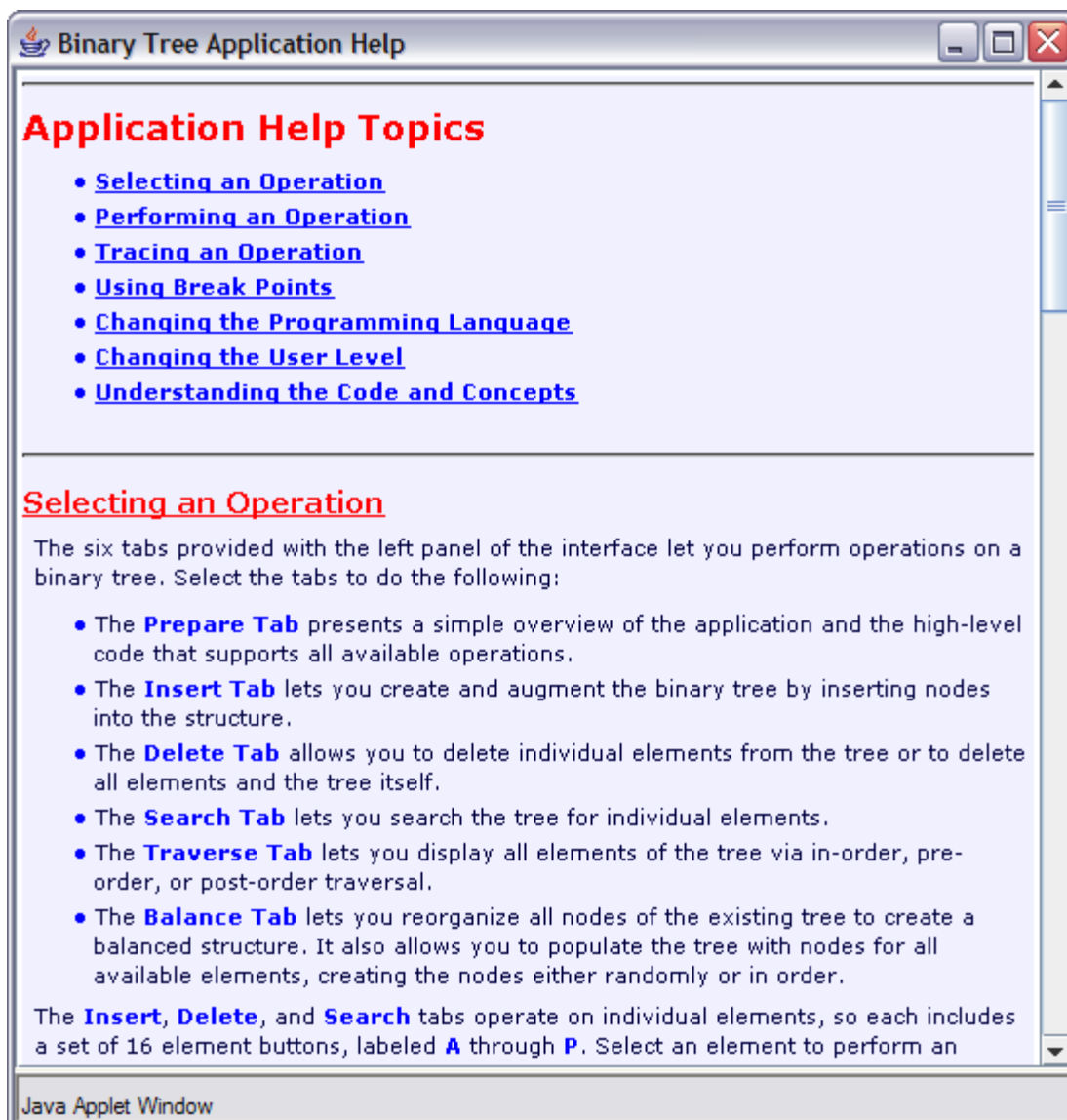


Figure 27: Help Documentation from Binary Tree

The most obvious means of selecting conceptual or help documentation is via the Concepts and Help buttons, respectively, of the applet (neither of which is shown here). These are always available from the upper right of the interface. The conceptual documentation is context sensitive, so clicking the Concepts button always displays the text associated with the tab currently selected on the interface.

Conceptual documentation is also accessible by clicking on the underlined name of a function or method from any of the code panes; the text that's displayed corresponds to the function or method on which the user clicks. With the concepts and help text windows, hyperlinking enables the user to jump from one area of the text to another, and all concepts or help text is always available from the respective window.

2.13 Introductory Selection Page

The final feature described is the introductory selection page for the applets, as shown in Figure 28. From this page, the user can choose the data structure, programming language, and user level with which the applet is to start. As described previously, the programming language and user level can always be changed from a running applet, but the user must return to the selection page to access the applet for the other type of data structure. The selection page also provides an overview of the applets.

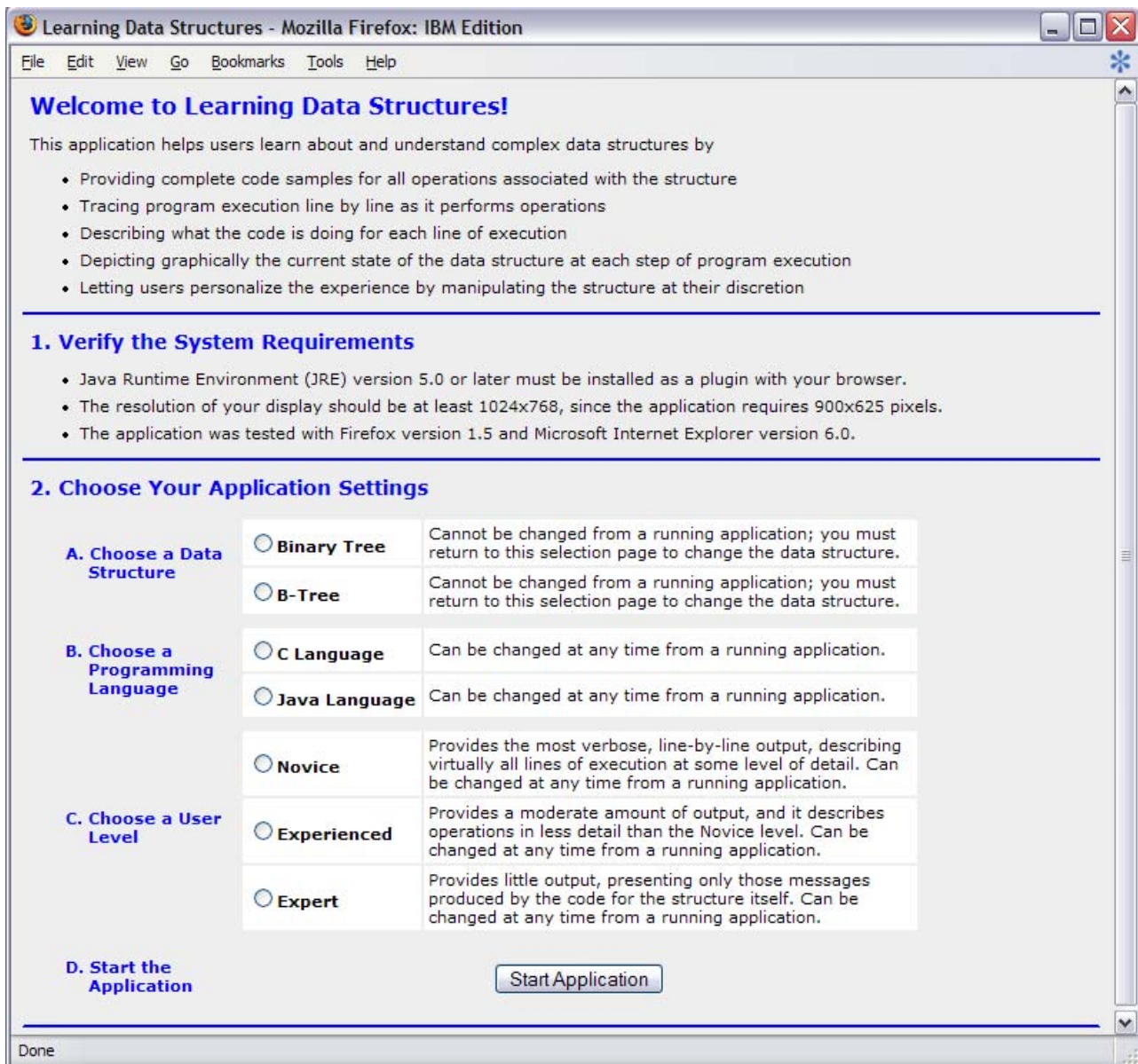


Figure 28: Introductory Selection Page

Note that this version of the page requires JavaScript to be enabled in the user's browser. If JavaScript is not enabled, the user is presented with a simpler version of the page that provides access to the two applets. In this case, each applet always starts with the C programming language and Novice user level selected.

3. Results of User Testing

User testing was conducted with members of Dr. Flynn’s graduate and undergraduate data structures classes at the University of Pittsburgh. Each user received a brief introduction to the applets at one of Dr. Flynn’s classes. The overview introduced the applets and their basic features very briefly, so no user was given extensive instruction regarding use of the applets or any personalized training. Beyond the brief introduction, subjects were asked only to use the applets to complement their classroom instruction, with no guidance about how they should approach or structure their usage.

Following use of the applets, each subject was asked to complete a questionnaire to provide feedback on their experience. Users could complete a Web- or Microsoft Word-based version of the questionnaire at their discretion. (See Appendix A for a complete copy of the questionnaire.) The following sections describe the users who completed questionnaires and summarize the feedback they provided.

Note that the versions of the applets tested by the users differed slightly from the versions described in Section 2. As tested, the applets had the following differences:

- **The Undo and Cancel buttons weren’t available during testing.** These were added late in the development cycle, after testing was completed.
- **The labels of the Next and All buttons read Next Step and All Steps, respectively.** This slight change was made to accommodate the new Undo and Cancel buttons.
- **The label and radio buttons associated with selection of user levels were not available.** Instead, users selected a Change User Level button to change the current level, and selection of that button was the only means of determining the current level.

Otherwise, the interface remained the same. (Unfortunately, I somehow neglected to ask a question about the utility of the user levels and their selection, so it’s unclear whether users found the previous version of the interface difficult to use.)

3.1 User Profiles

Test subjects were first asked to answer a few questions about their background. Table 3 summarizes the user profiles of the subjects who participated in the study. As mentioned earlier, all were students of data structures courses being taught by Dr. Flynn, and all were currently being taught binary trees and B-trees. Although few in number, the subjects did represent an interesting collection of users from the target audience (i.e., students learning about the data structures in question). All expressed their computer proficiency and programming knowledge as above average, so none should have experienced any difficulties understanding the basic thrust of the applets or their usage.

User	Sex	Age	Highest Level of Education Completed	Current Educational Program	Proficiency with Computers	Knowledge of Programming
1	F	>40	BS Chemical Engineering	MS Information Science	Above Average	Above Average
2	F	26-30	BA English Literature	MS Library and Information Science	Excellent	Above Average
3	M	21-25	High School	BS Information Science	Above Average	Above Average

Table 3: User Profiles

Subjects were also asked to describe their use of the applets. Table 4 describes the primary aspects of the applets with which each user experimented. As the table indicates, subjects used a fairly broad range of basic data structures, programming languages, and user levels. So the feedback should tend to represent at least some usage of all of the primary aspects of the applets.

User	Data Structures Tried	Primary Data Structure Used	Languages Tried	Primary Language Used	User Levels Tried	Primary User Level Used
1	Binary Tree B-Tree	Binary Tree	C Java	Java	Experienced	Experienced
2	Binary Tree B-Tree	B-Tree	C	C	Novice Experienced	Novice
3	Binary Tree B-Tree	B-Tree	C Java	C	Novice Experienced Expert	Novice

Table 4: Applet Usage

3.2 Scaled Test Results

This section presents and briefly discusses the results of the testing. Each user responded to a set of 18 Likert-scaled questions to describe their impressions of the applets. Users rated their agreement with each question using the following scale:

- 5 Strongly Agree
- 4 Agree
- 3 Neutral
- 2 Disagree
- 1 Strongly Disagree

In the figures that follow, the number shown for each question is the average response for the three users. The horizontal error bars represent the standard deviation from the mean. Figure 29 indicates users' opinions about basic high-level features of the interface. All features received a rating of at least 4.33, indicating a generally high level of satisfaction. Navigation of the interface and the helpfulness and clarity of descriptive messages received the lowest marks, but even they were not rated poorly. It's possible that the low rating for navigation may be the result of the inefficient means of identifying and changing the user level; the clearly visible radio buttons described in Section 2 were not available in the version of the interface that was tested.

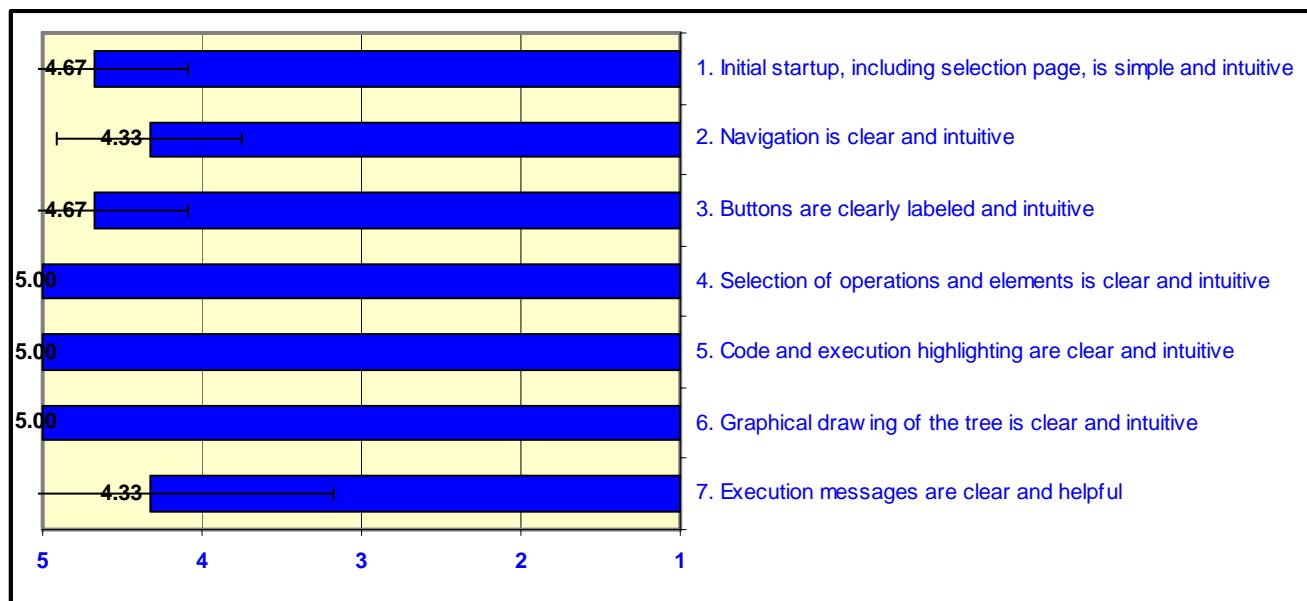


Figure 29: Responses to Questions about Basic High-Level Features

Figure 30 shows users' opinions of specific features of the applets. Because no user was required to try all features, users were allowed to indicate that they had not used a specific feature. For these questions,

the number of respondents is shown after the text of the question. Here again, all features received an average rating of at least 4.33, and only the clarity and helpfulness of All Steps mode received that rating. All other features were rated at least 4.5, indicating a high level of satisfaction in general.

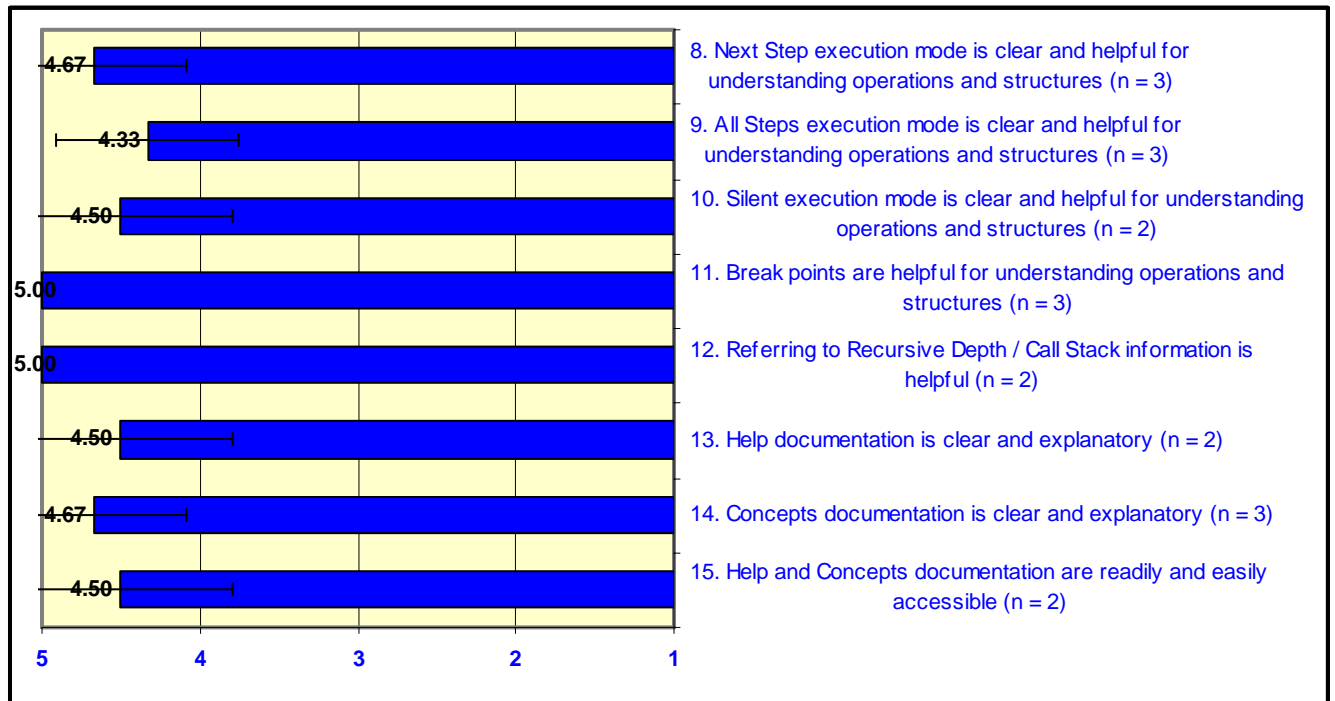


Figure 30: Responses to Questions about Individual Features

Finally, Figure 31 provides users' overall impressions of the interface, how well the applets helped them understand the data structures and their code and operations, and their overall impressions of the applets. For all three questions, the applets received perfect scores, indicating that users were generally quite pleased with the interface and utility of the applets.

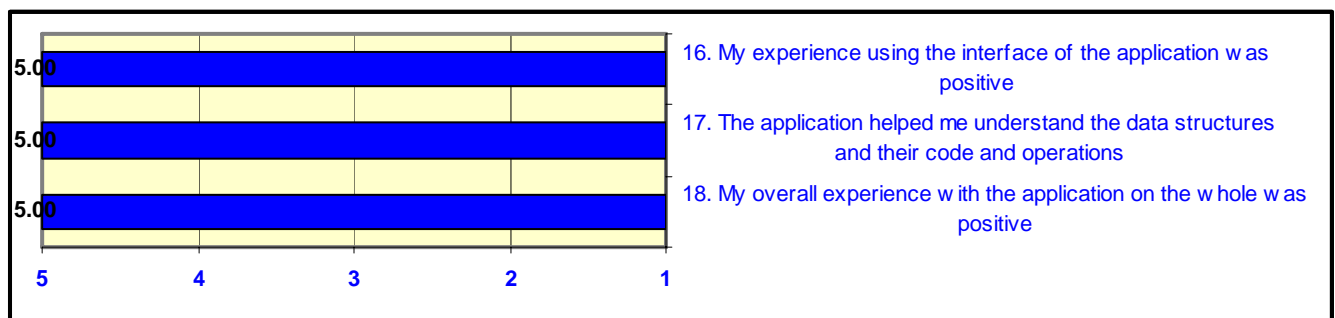


Figure 31: Responses to Questions about Overall Impressions

Based on this feedback, the only area where changes might be worth considering is the clarity of the descriptive execution messages. A future goal of the project might involve more specific evaluation and critique of the messages to ensure their appropriateness and explanatory qualities. Otherwise, the responses from these few users would appear to indicate that the applets were very well received.

3.3 Open-Ended Comments

The final section of the questionnaire asked users to comment on the applets in their own words. Users were presented with five questions to initiate responses. Most provided some level of meaningful feedback, which Table 5 captures.

1. Describe your overall impressions of the application and its interface.	
User 1	It is excellent.
User 2	My overwhelming impression of the application was favorable. In a strict graphic design sense, I feel it could be a bit more attractive – softer color palette, fewer angles, more diverse color selection – but this is the web designer in me speaking. As for usability, I was perfectly happy with both the application and the user interface, and found the layout to be easily viewed and used.
User 3	Everything is easy to use... self-explanatory... overall very clean and educational.
2. What did you like most about the application?	
User 1	Graphical presentation of the data structure was very helpful.
User 2	I appreciated the choices between novice/experienced/expert – I am new to data structures, and was able to use the ‘novice’ setting to understand how the code worked, then move through ‘experienced’ and ‘expert’ as I became more comfortable (I’m certainly no expert, but it gave me a chance to see the code in action!).
User 3	I love that you can hit the “all steps” button and watch the trees form while watching the code.
3. What did you like least about the application?	
User 1	Probably understanding the difference between Next Step, All Steps, and Silent.
User 2	Again, I will echo my answer to #1 – I was not wild about the graphic design. It does the job, certainly, in cleanly presenting the application, but a few tweaks here and there (color, graphics, reduction of angular borders) might enhance the user experience in subtle ways.
User 3	I talked about this in class -> No replay feature, but this isn't really all that necessary.
4. What suggestions do you have for improving the application?	
User 1	That would be hard to do. I think you have thought of every contingency.
User 2	Well, I was initially going to say that I would have liked some more extensive code documentation, but then I remembered that the function names were also hyperlinks – upon clicking on the link, I found all the information I needed. Perhaps a note somewhere on the page to the effect of “Function names can be clicked on for more extensive documentation” will lead more users to this information.
User 3	N/A
Briefly describe any problems or defects you encountered while using the application.	
User 1	I did not experience any.
User 2	Did not encounter defects or problems.
User 3	N/A

Table 5: Responses to Open-Ended Questions

The comments are interesting, but it's not clear that they suggest much in the way of future development for the applets. The comment regarding a replay feature will be addressed by the Undo button, once it's functional. The comments about the design of the interface are probably more one user's opinion than anything that really needs to be pursued, especially since no other user has yet to voice such issues, and even the user who made the comments stated that the design certainly presented the applets and their functionality cleanly. Otherwise, there's really not much to be said.

4. Conclusions and Future Directions

As with the first version of the applet created for INFSCI 2470, the latest versions of the applets received uniformly high marks from all users. Overall impressions of the interface and its usefulness as an educational tool were extremely high, a positive indication that the applets are headed in the right direction.

A slightly more significant discussion concerns future directions and a possible follow-on independent study. From a development perspective, four enhancements are worth considering:

- **Implement the Undo button to allow the user to undo the previous step of an operation.** The basic infrastructure for the button is in place, but none of the functionality exists at present. This is mandatory for future versions of the applets.
- **Rework the balance code of the binary tree.** Dr. Flynn expressed an interest in demonstrating a more elegant balancing algorithm for the binary tree. If a follow-on project is undertaken, I will discuss this possible change with Dr. Flynn.
- **Revisit the user messages to ensure they're as clear and descriptive as possible.** It's unclear how much can be accomplished in this area or where I would focus, given the limited feedback from the user testing, but it is an area that may call for further exploration.
- **Consider further code reorganization and modularization.** If it's thought worthwhile, a thorough reworking of the code might allow new data structures to be added in a more pluggable fashion. At a minimum, it would also allow the separate code bases for the two applets to be combined. This work is probably only of interest if additional data structures are to be supported or if extensive changes to the existing code prove necessary for some reason.

From the perspective of a potential independent study, a number of things would need to occur:

- **Conduct more extensive and thorough testing.** In addition to questions focused on ascertaining opinions about different aspects of the interface, a very useful undertaking would be to instrument the code to capture user keystrokes and general usage activity. This would help to understand how users are using the applets and identify possible difficulties. It would also lend itself to development of a more complete summary of the applets and their usage.
- **Research similar tools and interfaces to understand their features and usage.** Compare the applets to similar projects and research efforts to develop a better understanding of the benefits of the applets and other research efforts conducted in this area.
- **Write a formal paper for submission to a conference or journal.** For consideration in appropriate forums, the previous items need to be addressed and considered at a very detailed level. The Undo function is critical to the success of the applets, so that needs to be done as a matter of course. But the extensive user testing must be focused toward developing a better understanding of the applets, leading to a more robust and more thoroughly quantifiable discussion of their usage and benefits.

I will discuss these future directions with Drs. Brusilovsky and Flynn in the coming months to determine whether we can proceed with a second study. If so, I will work on the effort during the spring and, primarily, summer semesters of 2007.

A. User Questionnaire

Users who evaluated the applets and provided the feedback summarized in Section 3, “Results of User Testing,” had the option of completing a Web-based questionnaire or an equivalent questionnaire made available in Microsoft Word. The Web-based questionnaire required that JavaScript be enabled in the user’s browser; if it was not enabled, the HTML page defaulted to presenting the Word version of the questionnaire. Any user could also elect to complete the Word version of the questionnaire instead of the online forms, if they so desired, but all three subjects chose to complete the JavaScript version.

The two versions of the questionnaire were completely equivalent in all regards. The advantages of the Web-based form were ease of use and response validation. In terms of ease of use, the user could select a radio button or a checkbox to respond to most questions, the questions were grouped into multiple navigable pages for presentation purposes, and submission required just the click of a button. For response validation, the user could not progress from one page of the questionnaire to the next until he or she had provided valid responses to all questions. In cases where one question became valid based on the response to another question, the JavaScript code was intelligent enough to take such conditions into consideration and respond appropriately.

The following pages provide a copy of the three-page Microsoft Word-based questionnaire.