CSTutor: A Pen-Based Tutor for Data Structure Visualization

Sarah Buchanan
University of Central Florida
4000 Central Florida Blvd.
Orlando, Fl 32816
sarahb@cs.ucf.edu

Brandon Ochs
University of Central Florida
4000 Central Florida Blvd.
Orlando, Fl 32816
brandonochs@knights.ucf.edu

Joseph J. LaViola Jr.
University of Central Florida
4000 Central Florida Blvd.
Orlando, Fl 32816
jjl@eecs.ucf.edu

ABSTRACT

We present CSTutor, a sketch-based interface designed to help students understand data structures. It currently supports Linked Lists, Binary Search Trees, AVL Trees, and Heaps, and creates an environment in which a user's sketched diagram and code are combined seamlessly. In each of the data structure modes, the user can naturally sketch a data structure on the canvas just as they would on the white board. CSTutor analyzes the user's diagrams in real time, and automatically generates code in a separate code view to reflect any changes the user has made. Additionally, the code can also be edited and any new code changes animate the data structure drawn on the canvas. The connection between the data structure drawn on the canvas and the code implementation is intended to bridge the gap between the conceptual diagram of a data structure and the actual implementation. We also present the results of a perceived usefulness survey. The results of the study indicate that the majority of students would find CSTutor helpful for learning data structures.

Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer-assisted instruction (CAI)

General Terms

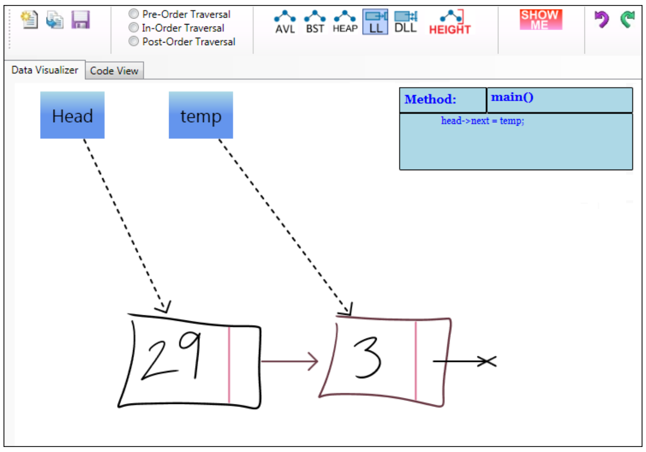
Algorithms, Human Factors, Experimentation

Keywords

Sketch-based input, data structures, visualization, animation

1. INTRODUCTION

Instructors in Computer Science generally present new data structures using Powerpoint presentations or drawings on a whiteboard combined with pseudo-code. Usually an instructor will have to draw several instances of a diagram to explain an operation. This can be troublesome for multiple reasons. First, students may not understand which piece of pseudo-code pertains to which change in the diagram. Second, it is often time consuming and difficult for instructors to draw each instance of the diagram. We designed CSTutor (see Figure 1) with the goal of bridging the gap between the conceptual diagram of a data structure and the actual implementation.



```
head->next = temp;
```

Figure 1: A screenshot of the sketching area in Linked List mode after nodes 29 and 3 are added.

In addition to diagrams, algorithm visualization tools are used by students and instructors as a supplemental learning aid. However, studies have shown that algorithm visualization alone does not improve a student's level of learning; it is the level of student engagement with the algorithm visualization that is effective [5]. In addition, many existing systems are not practical for use in the classroom or by students because creating the data structures and animations is too time consuming. These systems are not flexible enough to support creating and editing examples on the fly.

To facilitate a more unified approach to learning and to address the issues above, we developed CSTutor, a novel, pen-based application for data structure visualization that combines sketching, animating and coding of data structures. The intended users of CSTutor are students in a beginning data structures and algorithms class (CS1 at the University of Central Florida). The system lets the user sketch elements of a data structure and perform operations on them through the recognition of handwritten symbols and gestures. Once a gesture is recognized the operation on

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE'12, February 29–March 3, 2012, Raleigh, North Carolina, USA.
Copyright 2012 ACM 978-1-4503-1098-7/12/02 ...$10.00.

the data structure is animated while simultaneously adding automatically generated code to the code view. In addition, the code can be edited and any new code changes animate the drawing in real time. In other words, if the user sketches changes to the diagram, the code will dynamically change, and if the user edits the code, the diagram will animate and update itself. The data structures represented in CSTutor (Linked List, Doubly Linked List, Binary Search Tree, Heap, and AVL Tree) were chosen specifically because they are the most basic data structures taught in introductory computer science classes.

In order to gauge the usefulness of CSTutor in aiding CS1 students before incorporating this tool into the curriculum, we conducted a perceived usefulness survey. The tool was demonstrated to 88 students at the end of their CS1 class as a way to review the data structures before the final exam. A video demonstrating how CSTutor can be used to teach data structures can be found here: `http://youtu.be/uEja_cxrGCg`.

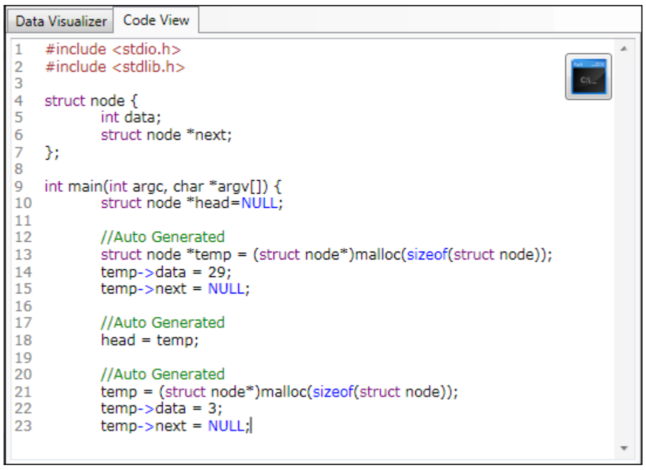
2. RELATED WORK

Many data structure visualization and animation tools have been developed over the past 30 years, without gaining much success in the classroom. Of these tools, Tango and XTango were some of the earliest algorithm animation systems that became popular in the early 1990's. XTango was similar in nature to what CSTutor accomplishes in that it attempts to aid computer science students in understanding algorithms through animation [12]. The difference with CSTutor, however, is that we created an environment where users can quickly and easily create their own data structures by sketching them. Additionally, CSTutor lets users create custom functions to manipulate the data structure and watch how their code interacts with it.

Two of the most widely used software visualization tools in use today are JGrasp and Jeliot, which focus heavily on debugging code rather than understanding the concepts of data structures. JGrasp is a comprehensive IDE that generates visualizations as needed from the user's actual program during routine development [2]. It differs from our system in that it focuses on creating visualizations from the user's code, while we create code from a sketch-based diagram and modify the diagram with edits made to the code. Jeliot [8] is similar to JGrasp, however it focuses on beginning concepts such as expression evaluation and assignment of variables, whereas our system focuses on core data structures such as trees and linked lists.

JHAVEPOP [4] is another teaching tool which focuses solely on linked lists by presenting specific linked list problems to solve, and then graphically displaying their results. The focus of this program is very similar to CSTutor, however we wanted to take this idea further by creating a system that can be used to teach users about multiple data structures with an interactive pen-based interface.

In addition, JAWAA and TRAKLA are algorithm animation tools that have had positive results in the classroom. JAWAA is a web-based algorithm animation scripting language that allows students to print animation commands from their own programs [1]. In contrast to CSTutor, JAWAA requires the students to have knowledge of what they want the animation to do, whereas CSTutor creates the animations behind the scenes and does not require the user to enter any special print commands. With TRAKLA



```
#include <stdio.h>
#include <stdlib.h>

struct node {
        int data;
        struct node *next;
};

int main(int argc, char *argv[]) {
        struct node *head=NULL;

        //Auto Generated
        struct node *temp = (struct node*)malloc(sizeof(struct node));
        temp->data = 29;
        temp->next = NULL;

        //Auto Generated
        head = temp;

        //Auto Generated
        temp = (struct node*)malloc(sizeof(struct node));
        temp->data = 3;
        temp->next = NULL;
```

Figure 2: The code area in Linked List mode after nodes 29 and 3 are added to the list.

students can solve algorithm problems with a drag and drop interface and then are provided automatic assessment [7]. CSTutor differs from TRAKLA since it has an interactive pen-based interface as well as a source code component.

There have been several pen-based tools developed for learning and entering information in the computer science domain. SketchUML and Tahuti are sketch-based tools for UML class diagrams [6, 10]. GraphPad is a web-based software system that lets the instructor and students interact electronically during class on Tablet PCs. Students can also submit solutions to graph problems and receive feedback [9]. CSTutor is different from GraphPad in that GraphPad does not incorporate the code implementation of the data structure and does not use animation, whereas CSTutor does both. The system presented in [14] is a pen-based flowchart recognition system that recognizes flowchart components and generates runnable C code. This is similar to CSTutor in that it recognizes ink and then generates code, however it is not applicable to data structures and does not reflect changes to the code in any way in the sketched diagram. To the best of our knowledge, CSTutor is the first pen-based application that incorporates animation and coding, designed for aiding Computer Science students in understanding data structures.

3. CSTUTOR USER INTERFACE

CSTutor has two main views that users are able to switch between. The default view is a canvas (see Figure 1) which has been designed to look and feel like pen and paper that provides feedback in an unobtrusive way. The buttons on the screen are minimalistic, and all of the commands are given through gestures where possible. Running alongside this is the code view area, which gives the user a scaffold since any sketched data structure generates corresponding code, giving students a head start in writing various data structure operations. The code view behaves just like a typical IDE by highlighting keywords and providing compile and debug information. However what sets CSTutor apart from a typical compiler is that any updates to the code will actually impact the data structure in the canvas, and the user can step through the animation to see how their code impacts the data structure.

The sketching area for CSTutor is built on Microsoft's InkCanvas, and uses both gesture and character recognition. Gestures are detected by using the iStraw algorithm [13] for cusp detection in combination with heuristics defining each gesture. We found this to be a significant improvement over Microsoft's built in gesture recognizer, which was not only inaccurate at times but also limited us to use a pre-defined gesture set which was not appropriate for our application. We aim to model the gestures after what is normally done on pen and paper. For instance, circling a value when sketching a Binary Search Tree node is what is normally done on pen and paper, thus the completion of the circle gesture is what alerts the system that a new node has been added to the tree.

3.1 Sketching Area

The user is able to choose between the different data structures, which allows us to limit the available gestures to only what operations are available for the current data structure. In order to keep the UI as natural as possible, a combination of several basic gestures are used for each interaction mode. Linked Lists and Doubly Linked Lists use a combination of rectangle, line, and scribble erase gestures. For the trees (Binary Search Tree, Heap and AVL Tree) a combination of circle, arrow, tap, and scribble erase gestures are used. All of the gestural operations that we have chosen are based on how a professor would teach these concepts using a diagram, and the content comes from a variety of course lectures and textbooks [11].

Additional pointers to nodes are represented as blue rectangles on the canvas. The head pointer remains static for each different data structure to emphasize that it is necessary to have a pointer to the beginning. The temp pointer box is also always there as a temporary pointer to any new nodes, or nodes that are being modified. The user can use a line gesture to connect any of these pointers to different nodes. For example, changing which node the head points to is useful for inserting a node to the front of a linked list, and changing which node the temp pointer is pointed to can be useful for deleting a node in these data structures. In addition, any pointers the user creates in their code appear on the canvas as blue rectangles with the corresponding variable names.

3.2 Code Area

Any sketch created in the canvas always creates a section of code in order to show the connection between the diagram and any code changes. When a sketch is completed, the corresponding code is simultaneously added to the main method in the code area (see Figure 2) and is also shown in an information box in the user's sketch area (see Figure 1). The auto-generated code is meant to demonstrate how memory works, and is not the modular code you would see in a program. Users can edit the auto-generated code, add their own functions and operations, compile and run the code, and step through the resulting changes to the diagram. Compiler output is also displayed in the user's code area, and any warning or error messages are shown.

The Show Me button is important for animating the user's compiled code and is used to iterate through the animations step by step. Each step in the animation has the potential to create new nodes, delete existing nodes, or move the connections of any of the nodes or pointers.

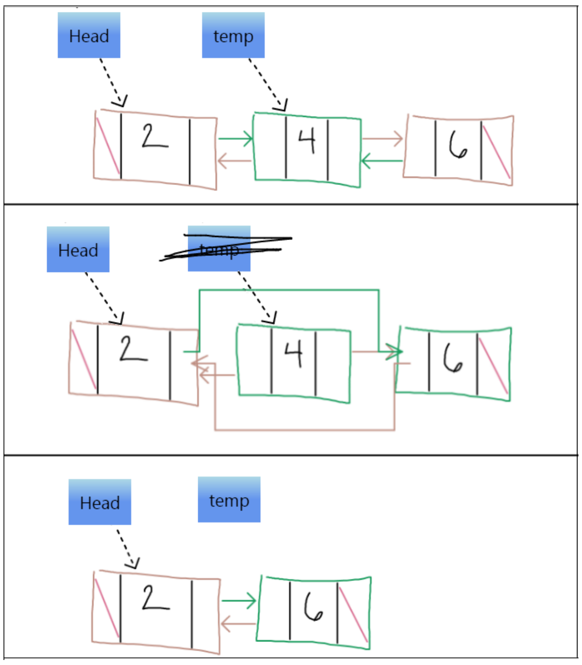


Figure 3: The steps required to delete a node from a Doubly Linked List: (1) Assign temp to the node to be deleted, (2) bypass the node to be deleted, (3) free the node pointed to by temp.

3.3 Interaction Modes

3.3.1 Linked List and Doubly Linked List

The sketching area is designed to allow the user to do the same operations on the data structure through sketching as they can through code. They can create nodes, connect and disconnect nodes, assign/un-assign pointers to nodes, delete nodes and pointers, and update the data and next values. For example, a linked list node is created by writing the value of the node and then boxing that region in with a rectangle. When the rectangle gesture is completed, the surrounded strokes are sent to an online recognizer that analyzes the user's handwriting to determine the numerical value. After the node is drawn it is translated and scaled to make room for subsequent sketches. Then, a small box is drawn within the node to represent the allocated memory for a pointer to the next node. A null pointer stroke is also added to this memory field for nodes where the next link is null (see Figure 1). Sketching a node on the canvas also affects the code in the code window, and generates code that does the following: (1) allocates a node in memory, (2) assigns a value to that node, and (3) sets the node's next value to null.

For operations such as delete, instead of the user simply scratching out a node we enforce that the user goes through each step they would have to implement in code using gestures in the canvas (see Figure 3). For example, the temp pointer first needs to be assigned to the node being deleted by drawing a line to the node from the temp pointer box. Secondly, the neighboring nodes need to be linked to each other to bypass the deleted node. Finally, the node pointed to by temp can be freed by using a scratch out gesture on the temp node and the node is deleted from memory. These steps would create the following code:

```
temp = head->next;
head->next = head->next->next;
head->next->next->prev = head;
free(temp);
```

We designed the Linked List and Doubly Linked List interaction modes to require that the lists be connected, or at least that all nodes are accessible by a pointer, to demonstrate memory leaks to the user. For example, if a user creates their first node without connecting it to the temp or head pointer, any future nodes will not be added and the user is shown a warning message. Additionally, in the delete example above, if the user bypasses the node they wish to delete without first assigning temp to that node, then temp is automatically assigned to that node. This rule is presented in [11], and says that access must be preserved to all nodes that will be needed later in the solution of a problem.

3.3.2 Binary Search Trees

Tree nodes are created by writing the value of the node and then circling that number, which sends the surrounded strokes to an online recognizer, just as with the Linked List mode. After the node is drawn it is dynamically arranged on the canvas and lines are drawn to each child node if the children are not null (see Figure 4). Similar to the Linked List mode, sketching a new node with a value of 5 creates the following code:

```
temp =
(struct node*)malloc(sizeof(struct node));
temp->data = 5;
temp->left = NULL;
temp->right = NULL;
head = insert(head, temp);
```

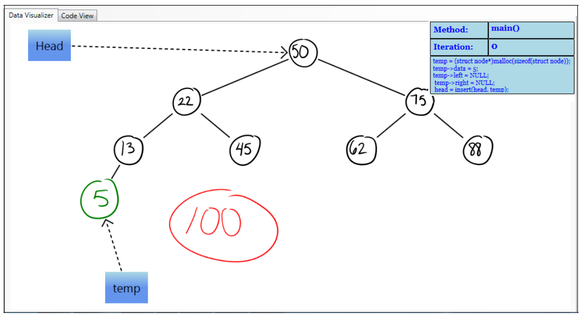
Note that since inserting into the tree requires more than a single line of code, we have abstracted the insert code into its own function and we hide it from the user's code window. We chose to hide the implementation of insert from the user so that they can code their own insert function and visualize it on the canvas.

We initially supported having a newly created node moved to the correct position in the tree automatically. However, we want to ensure that students know how to insert a node according to the binary search tree property, so we make sure the user sketches the node in the proper area of the tree. If the node is added to the correct position, it is colored green. Otherwise, if the node is in an incorrect position the strokes are colored red and the user cannot make additional operations on the tree until the offending node is scribbled out (see Figure 4).

In addition to inserting new nodes in the tree, the user can delete a node by scratching out the node. Since insertion and deletion are the main operations made to a binary search tree, gestural operations (e.g., circle for creating nodes and scribble erase for deleting nodes) are provided for them and any other operations on the tree must be implemented in the user's code.

3.3.3 Heaps

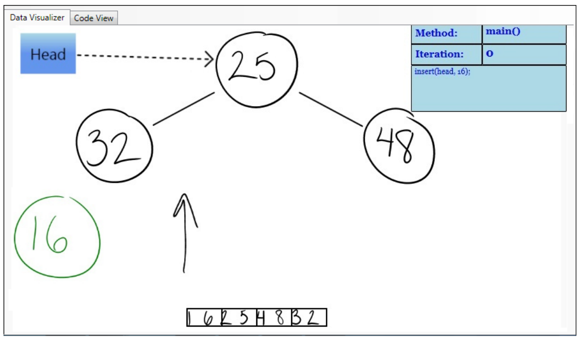
The heap data structure we support is a minimum binary heap, since this data structure is taught in entry level Computer Science courses. The heap is displayed in the same manner as the Binary Search Tree, with an additional array



Method:	main()
Iteration:	0

```
temp = (struct node*)malloc(sizeof(struct node));
temp->data = 5;
temp->left = NULL;
temp->right = NULL;
head = insert(head, temp);
```

Figure 4: The sketching area in Binary Search Tree mode. Node 5 was added to the correct location and is colored green, whereas node 100 was added to the incorrect location and it is colored red.



Method:	main()
Iteration:	0

```
insert(head, 16);
```

Figure 5: The sketching area in heap mode. Node 16 has just been added to the heap, and the up arrow gesture has been performed to swap 16 with its parent node. Beneath the heap the current state of the array is shown.

underneath that is updated after each operation. Although a binary heap is usually visualized as a tree, it is generally implemented using an array [11], thus it is necessary to show the heap and array at the same time (see Figure 5).

The main operations performed on a heap are inserting new nodes and removing the minimum element. Since there are many steps for the insert and remove minimum operations, we have abstracted both into their own functions, just like the insert operation in Binary Search Tree mode.

Similar to the Binary Search Tree mode, if the node is added to the correct spot (the next open leaf node), the node is colored green, otherwise it is red. If the node that is inserted is less than its parent, then a percolate up function is recursively called, so that the node can continue to swap locations with its parent until it is in its correct location in the tree. Originally, the swap operations were animated immediately, so the node would be swapped up the tree without any user input. However, since many exams ask the student to show each step in an insert heap operation, we make the user draw an up arrow gesture for each necessary swap up, and a down arrow gesture for each swap down (see Figure 5).

The second operation supported in heap mode is to remove the minimum node, which is done with a scribble erase gesture on the root of the tree. Once this node is removed, it must be replaced with the last leaf node in the tree. Since we want to ensure that the user knows which node to select as a replacement, we require a tap gesture next to the replacement node. If the wrong node is selected it is colored red, and the program will wait until the correct node is selected before finally animating the swap. Once the root is replaced, it must be swapped with its children until the root becomes the minimum element in the tree if it is not already. After each operation on the heap, the array underneath the tree is updated to reflect the current state of the heap data structure as it exists in memory.

3.3.4 AVL Trees

Many students often struggle with understanding the conceptual operations on AVL trees as well as their implementation. One common approach to teaching AVL trees is to teach the conceptual operations on an AVL tree diagram before delving into the code implementation. Some examples of conceptual operations that are taught are identifying the balance factor, determining if a rotation is needed, determining what kind of rotation is needed, and finally what the tree should look like after it has been rotated. In a classroom setting all of these operations are initially presented without the code and are still difficult to understand. Once the code is presented many students become even more confused with the recursive implementation. CSTutor adds guidance to help with understanding conceptual operations and delving into the code.

When a user is sketching an AVL tree in CSTutor and a rotation is needed, the user cannot continue sketching until the rotation is completed. A single rotation is completed with an arrow gesture in the required direction, and once the correct gesture is done the tree animates to reflect the changes of the rotation. Since a double rotation is simply a call to two consecutive single rotations, we enforce that the user makes two arrow gestures for a double rotation and the user can visualize each single rotation separately to see specifically what the code is doing. We designed CSTutor to force students to actively rotate the tree with arrow gestures, instead of allowing users to passively observe the AVL tree animations to promote active learning.

4. INITIAL ASSESSMENT

Before incorporating CSTutor into the curriculum for a semester long user study, we demonstrated its capabilities to a CS1 class. CSTutor was presented to 88 students prior to the final exam. Approximately 30 minutes were dedicated to showing review problems for each of the following data structures: Linked Lists, Binary Search Trees, Minimum Heaps, and AVL Trees, followed by 5 minutes for questions and 10 minutes to complete the questionnaire. The students' feedback was based off of this review which covered two examples of how to use each data structure, focusing on both diagram and code changes. Students were also shown how their code changes could be animated and traced through step by step in the sketching view. The review was conducted using an HP tc4400 Tablet PC connected to a projector. After the review session we asked the students to assess the perceived usefulness [3] of CSTutor, and to think about how it could have impacted their ability to learn the course material. The questionnaire consisted of 10 statements on a Likert scale with options: (1) Strongly Disagree, through (7) Strongly Agree.

The results of the survey are shown in Figure 6 and indicate that students found that CSTutor would be generally useful in learning data structures. Questions 7 through 10 show that students felt CSTutor would be useful for the variety of data structures presented, but did not prefer one over another. Question 6 had the lowest average, indicating that students did not feel as strongly about CSTutor helping to find bugs compared to its other features.

We also asked two open response questions: (1) "Do you feel that using CSTutor would be useful in better understanding how some data structures (i.e. Linked Lists, Binary Search Trees, Heaps, and AVL Trees) are implemented? If so, how? If not, why not", and (2) "What changes would you like to see made to CSTutor to make it better?"

The answers to the open response questions were also positive. Most students (61 students) commented that visualizing operations on the data structures would help them. For example, some students said that: "Seeing the code visually with the animation makes connecting a concept to code a lot easier", "I think it would be very useful for AVL trees, because rotations are hard to visualize", "As a visual learner,

Table 1: The CSTutor Perceived Usefulness questions given to students.

	Student Assessment Questions
Q1	CSTutor would have helped me with the material in CS1.
Q2	CSTutor would help me understand how each step in a program is working.
Q3	CSTutor would enable me to complete my homework assignments more quickly.
Q4	Animating what I'm coding would help me understand data structures better.
Q5	Auto-generating code while sketching a diagram would help me understand the relationship between the diagram and the code.
Q6	CSTutor would help me find errors in a program.
Q7	CSTutor would help me understand the implementation of Linked Lists better.
Q8	CSTutor would help me understand the implementation of Binary Search Trees better.
Q9	CSTutor would help me understand the implementation of Heaps better.
Q10	CSTutor would help me understand the implementation of AVL Trees better.

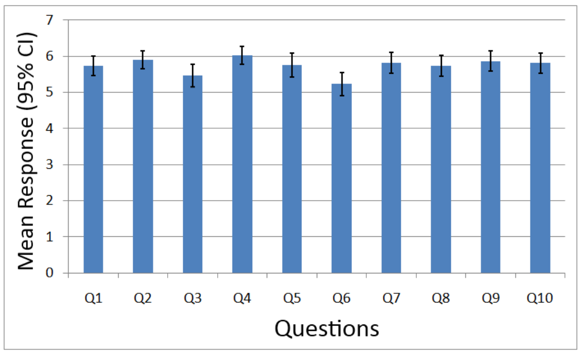


Figure 6: The average results of the Likert scale questions shown with the confidence intervals.

it is helpful for me to literally see what a function is doing. I feel it helps show why you need to do some things, like allocating memory, or making a temp pointer", and "Yes, being able to see the data structure in a visual, interactive way makes coming up with the actual code easier."

Four students had negative open response comments about CSTutor. For example, students commented "I don't really think this would be very useful. This would turn students into being lazy", "Sometimes you might not know what the software is doing", and "No, it mirrors the way it was taught. Therefore, it cannot help to better understand the subject."

Another four students felt that they understood the material enough without the tool, but that it might help other students who are visual learners. For example, one student commented "I feel it would help with generating code, but I don't find the material in CS1 too difficult. Would definitely help with intro to linked lists for someone with little knowledge of how pointers work." Ten students had neutral responses, and the remaining nine students left the open response blank.

There were also many useful suggestions in the second free response question. Some examples include, adding additional data types, giving more feedback for compiler or run-time errors, and more feedback if the user does something wrong.

5. FUTURE WORK

In the future we would like to make CSTutor more robust to user errors and provide more pedagogical feedback. Although users can visualize many logical mistakes by watching the animations of their functions, errors that cause the program to crash (such as infinite loops and referencing null pointers) are hard to determine without tracing through the code. We would like to allow the capability to more intelligently alert the user of these types of errors, instead of simply showing the compiler output. We would also like to give the user the option of using different programming languages, so that CSTutor can be used outside of CS1 at UCF. In addition, the current automatically generated code needs to be refined so that it demonstrates how memory works as well as proper programming abstraction. Ultimately we would like to incorporate all additional data structures and algorithms seen in CS1 and CS2. In the near future we are going to conduct a semester long user study in a CS1 class examining the effects of using CSTutor on students' understanding and learning of data structures.

6. CONCLUSION

We have presented CSTutor, a pen-based tool for dynamic visualization of data structures. CSTutor provides support for Linked Lists, Doubly Linked Lists, Binary Search Trees, AVL Trees and Heaps and visualizes their operations directly on the canvas. Users can input each data structure and operation using pen-based techniques and CSTutor recognizes the user's handwriting using character recognition. In addition, the user can input functions on the data structure in code and visualize their functions that operate on their sketches. We also conducted an informal user study, where we found the data structures and animations were well liked by the participants indicating formal semester long evaluations are required.

7. ACKNOWLEDGEMENTS

This work is supported in part by NSF CAREER award IIS-0845921 and NSF awards IIS-0856045 and CCF-1012056.

8. REFERENCES

[1] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S. H. Rodger. Jawaa: easy web-based animation from cs 0 to advanced cs courses. *SIGCSE Bull.*, 35:162–166, January 2003.

[2] J. H. Cross, II, T. D. Hendrix, J. Jain, and L. A. Barowski. Dynamic object viewers for data structures. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, SIGCSE '07, pages 4–8, New York, NY, USA, 2007. ACM.

[3] F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13:319–340, September 1989.

[4] D. Furcy. Jhavepop: visualizing linked-list operations in c++ and java. *J. Comput. Small Coll.*, 25:32–41, October 2009.

[5] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *Proceedings of the 2003 ACM symposium on Software visualization*, SoftVis '03, pages 87–94, New York, NY, USA, 2003. ACM.

[6] T. Hammond and R. Davis. Tahuti: a geometrical sketch recognition system for uml class diagrams. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[7] L. Malmi and A. Korhonen. Automatic feedback and resubmissions as learning aid. In *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*, pages 186 – 190, aug.-1 sept. 2004.

[8] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with jeliot 3. In *Proceedings of the working conference on Advanced visual interfaces*, AVI '04, pages 373–376, New York, NY, USA, 2004. ACM.

[9] R. P. Pargas and S. Bryfczynski. Using ink to expose students' thought processes in cs2/cs7. *SIGCSE Bull.*, 41:168–172, March 2009.

[10] L. Qiu. Sketchuml: The design of a sketch-based tool for uml class diagrams. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, ED-MEDIA, 2007.

[11] T. A. Standish. *Data Structures, Algorithms, and Software Principles in C.* Addison-Wesley, Reading, Massachusetts, 1995.

[12] J. Stasko. Animating algorithms with xtango. *SIGACT News*, 23:67–71, May 1992.

[13] Y. Xiong and J. J. LaViola, Jr. Revisiting shortstraw: improving corner finding in sketch-based interfaces. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 101–108, New York, NY, USA, 2009. ACM.

[14] Z. Yuan, H. Pan, and L. Zhang. A novel pen-based flowchart recognition system for programming teaching. In E. Leung, F. Wang, L. Miao, J. Zhao, and J. He, editors, *Advances in Blended Learning*, volume 5328 of *Lecture Notes in Computer Science*, pages 55–64. Springer Berlin / Heidelberg, 2008.