

# Data Structure Visualiztation as a Learning Tool

Tom Weiss-Lehman

December 11, 2005

## Contents

<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	The psychology . . . . .	2
2.2	Surveys and Studies . . . . .	2
<b>3</b>	<b>Existing Visualization Tools</b>	<b>2</b>
3.1	Brief History . . . . .	2
3.2	Swan Overview . . . . .	3
3.2.1	Evaluation . . . . .	3
<b>4</b>	<b>Project Overview</b>	<b>3</b>
4.1	User Interface . . . . .	3
4.2	Discrete Elements of the Project . . . . .	4
4.2.1	Display . . . . .	4
4.2.2	Meta Data . . . . .	4
4.2.3	Data Structures . . . . .	5
4.2.4	Supported Data Structures . . . . .	5
4.2.5	Programer Interface . . . . .	6
<b>5</b>	<b>Future Work</b>	<b>7</b>
5.1	Restructure the Code . . . . .	7
5.2	Input Parsing . . . . .	8
5.3	Time Control . . . . .	8
5.4	Algorithm Animation . . . . .	8
5.5	Effectiveness Study and Curriculum Integration . . . . .	8

## 1 Abstract

Research has found that most students learn faster and better with an interactive approach to lectures and instruction. I have developed an interactive tool to assist students with understanding some of the basic data structures used in computer science. The tool I have developed works well for linear data structures however fails for more complex tree structures. Several different options for correcting this problem are discussed under the Future Work section.

## 2 Background

### 2.1 The psychology

Programming is an activity requiring the use of both sides of the brain. Both logical and verbal thinking are required to successfully design and write software[5]. This has led to the development of algorithm and data structure visualization tools, which allow a student to process a concept 'in parallel', both visually and logically, helping the concept stick. [5] gives four reasons visual programming is stimulating:

1. Pictures are more powerful than words.
2. Pictures aid understanding and retention
3. Pictures provide incentive for learning
4. Pictures are understood regardless of language.

### 2.2 Surveys and Studies

Several informal surveys have shown that many computer science educators believe that visualization tools aid learning, however, professors mostly use these visualization tools in lectures to demonstrate concepts, as opposed to using the tool in a homework or lab setting[7]. Studies conducted on the effectiveness of visualization tools have obtained some interesting results. One of which is that the quality of the animation does not have any effect on the effectiveness of the tool as a learning device. Static frames work just as well as smooth animations [6]. More important than a smooth animation and visually appealing graphics are clear concise pictures containing all pertinent information and suppressing extraneous detail [7].

Several studies have found the key difference among visualization tools is the engagement of students. In one study, it was shown that students who passively view visualizations have a much lower rate of retention than students actively engaged in the visualization. Methods used to engage students currently consist of having students create data structures and populate them in labs, asking students questions about the state of the data structure after an upcoming operation, or asking students to solve problems at a high level using various data structures and algorithms provided. Despite their benefits, visualization tools are not built into most course curriculum's. Most professors cite the large time requirements of finding a tool, learning to use it, and developing examples that fit into their curriculum as reasons not to use them[7].

## 3 Existing Visualization Tools

### 3.1 Brief History

According to [5] there are many different tools for algorithm visualization. These tools have been evolving since the early 80's first beginning with "pretty-printing" and from there moving to tools like XTango, which displays a dynamic view of all variables and their current value along with the code currently being executed. Most of the tools that exist today are written in C/C++ and are designed for a specific platform. Also most of the tools that

I have looked at are designed to step through an algorithm and show the state of all the variables and the current command being executed. This tools however, generally do not display an view of data other then individual variable names. This makes a conceptual view of the data virtually impossible for any complex data structure.

## 3.2 Swan Overview

One tool I found that is very similar to the one I have developed is the Swan Visualization System [1]. Swan is a tool allowing users to visualize data structures and execution processes of a program. Swan allows either the logical or physical layout of a data structure to be viewed. Swan also allows the user to request changes to the data structure, thus creating a separate animation based on a previous one.

Swan operates by taking a C/C++ source file annotating the source file, producing source that Swan can run as an animation. This allows instructors to use previous examples with out having to re code them. Swan also can display some textual information about the data structure and the operations that are occurring on it.

### 3.2.1 Evaluation

Swan seems to be a good tool which provides an easy to use interface for the creation of animations. The interface for the student also seems to be clear and easy to understand. The downside to Swan is that it does not allow a user to directly change the animation. The user must rely on the Swan code processor to correctly annotate the code. Also, while Swan is designed to be portable to different environments, since it is written in C/C++, the source must be recompiled and possibly edited just to make the transition from one environment to another. This hampers portability because the executable must be recompiled for different platforms.

The tools discussed here are all available from different universities and have been used with success in different curriculum's, however there are several areas in which this project is different or provides improved functionality.

## 4 Project Overview

This project has two goals, one is to provide a set of ADT implementations and the visualization to those implementations for use as in class demonstrations or as out of class review work. The other goal of this project is to provide an extensible frame work for software visualization, allowing professors or students to create other animations of either data structures or algorithms. The project is written in Java to provide as much portability between environments as is possible with current technology.

### 4.1 User Interface

This project provides the user with the ability to execute commands on a data structure and view the results of those commands, to see how the data structure changes and how the information is stored. Users can create, destroy, and manipulate data structures through a command prompt, which is, syntactically, very similar to a modern programming language (e.g C, Java). Users also have the ability to save and load project files consisting of a history

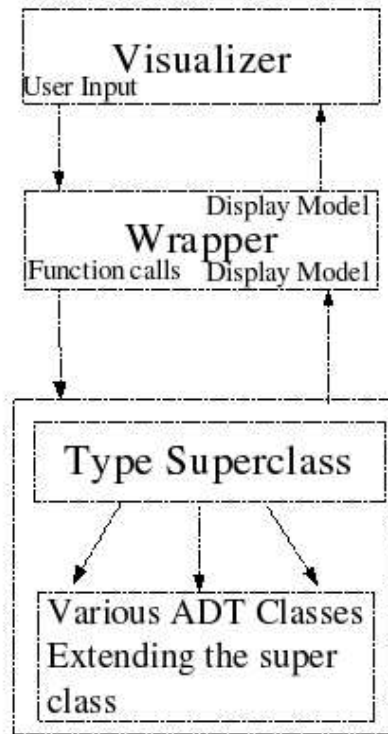


Figure 1: The layout of the three levels of this project and a description of the information passed between the levels.

of commands, which can be used either to provide students with a premade data structure or to turn in a lab assignment to a professor.

## 4.2 Discrete Elements of the Project

### 4.2.1 Display

The display element of this project is composed of two classes, Visualizer and ADTDisplay. Visualizer sets up the User Interface and handles getting user input. ADTDisplay is simply a wrapper class to allow for changing Data Objects during runtime. The user interface consists of a command line, with previous commands displayed below, a text area for messages to be passed from the data structure describing the current operation, an area for the return value from any operation to be displayed, a list of all data structures currently in memory, and a main window for the current data structure to be displayed in. (See Figure 2)

### 4.2.2 Meta Data

The next element, the meta data, is contained in a single class, ADTWrapper. The meta data for this tool is relatively simple. For each data structure, the structure its self is stored along with the name the user entered for the structure, which is a unique key for the structure, much like a variable in a programming language. This class also contains the code to parse user input and make appropriate function calls based on that input. This class also sends the appropriate display model to visualizer, calling functions from whichever ADT is currently

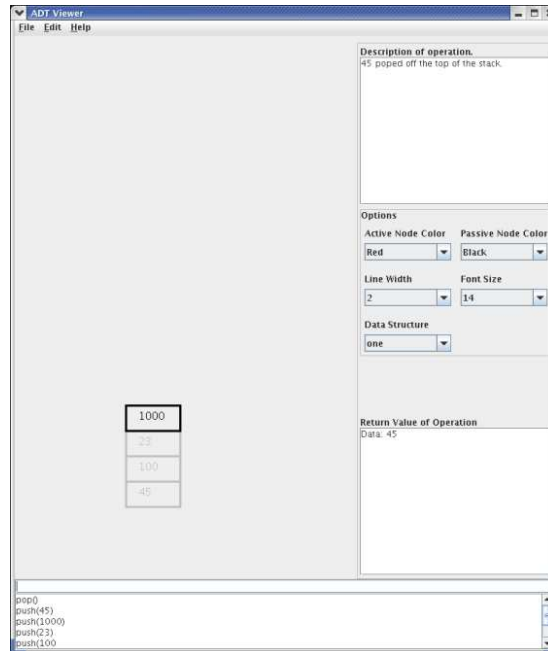


Figure 2: The Graphic User Interface and a Stack being worked with.

active. This allows Visualizer to have no notion of different ADT's and make all of its calls to a single structure.

### 4.2.3 Data Structures

The data structure level consists of one parent class, ADTType, and any number of child classes, all of which must extend ADTType. This provides a common function interface and allows the meta data level to store the data structures without knowing which types of data structures the user is going to instantiate until runtime. The parent class ADTType contains all the functions that can be called by the user and a paint function. ADTType's paint function attempts to draw the ADT based on its type. This is the point at which the design of this program shows its weakness. The only types of data structures currently able to be drawn are those that have a similar structure to a stack or queue. Other data structures are simply too complex to be drawn without more knowledge of the underlying structure (See Future work for more on this).

### 4.2.4 Supported Data Structures

Last In Last Out and First In First Out, stack type data structures are currently supported by the paint function. These data structures, which can easily be represented as a linearly structured array are the only data structures the current structure of this program will be able to display. To display more complicated data structures, the paint function must have more knowledge of the position of each node within the data structure. (See Future work)

### 4.2.5 Programmer Interface

This project is designed to be easily extended to include other data structures. Consequently, to extend this project, only a few steps are necessary. The first step is an implementation of the desired data structure which extends ADTType. ADTType defines several basic functions called by the command parser which are required to be compatible with the rest of the project. The basic data type passed to the data structure is always a String and that is required to be the return type for any function returning a value. However, Java provides simple conversion from any object to a string, and vice versa.

```
import edu.earlham.cs.CSLet.ADTType;
...
public class NameofADT extends ADTType {
    public NameofADT () {
        type = someType;
        fMap.put(dataTypeAdd, add);
        ...
    }
    public void add(String x) {
        //code to add x to the data structure
    }
    // other functions implemented here
    ...
}
```

After implementing the data structure, the command parser must be modified to create a new instance of the data structure when the user enters that command. This amounts to one more element in an if..then block.

```
public class ADTWrapper {
    ...
    public String parseCommand( String Command ) {
        ....
        if ( "new".equalsIgnoreCase( cmd[ 1 ] ) ) {
            if ( "stack".equalsIgnoreCase( cmd[ 2 ] ) ) {
                return addADT( cmd[0], new GStack() );
            } else if ( "NameofADT".equalsIgnoreCase( cmd[1] ) ) {
                return addADT( cmd[0], new NameofADT() );
            }
            ....
        }
    }
    ....
}
```

Here another advantage to using Java is demonstrated. Java provides a tool called javadoc, which parses specially formatted comments in Java source code, and creates a set of cross linked web pages with those comments describing classes functions and parameters.

This also automatically includes information about all the Java standard classes that are imported or extended by classes in the source tree, making understanding a program very easy.

## 5 Future Work

### 5.1 Restructure the Code

Since the current structure of this program does not allow display of other data types not currently supported the structure of this program needs to be redesigned. The problem with the current structure of the program is that individual data items have no concept of where they should be display, or of their position within the overall data structure. This means the program displaying the data structure must be able to guess where all the items in the structure are placed. For a linear structure, this is not difficult, but for a tree, for example, it rapidly becomes very difficult. Currently, I beleive the best way to correct this problem is to provide data structures with some ordering, implicit, or explicit.

**Links** The first idea I am considering is to require that a linked structure be used to store data. Then the layout of the data structure can be easily determined based on the number of links each data object has. There are of course still difficulties in determining the layout of complex structures, but I believe this approach will make them surmountable. The disadvantage to this approach is the data structure is required to implemented with links. This means that comparisions between different implementations of a single data structure are impossible, unless the programmer wants to maintain two instances of the data for a single structure.

**Structured Array** The second approach I am working with is to require the data structure to provide an array representation of the data with a known structure. If the branching factor is provided along with the array, the layout of the data could be determined. The advantage to this approach is it allows the person programming the data structure to use any implementation they want, allowing easy comparisons between different implementations. This disadvantage to this is it requires the programmer to implement a function to convert their data structure to an array, which could be very, very difficult depending on the data structure.

**Absolute Position** A third approach would be to require the programmer to provide an absolute position for each data object. This process could be abstracted slightly, breaking the display area up into cells and allowing the programmer to specify the row and column for each data object, and letting the display function translate between rows and columns and actual pixels. This approach is probably the most powerful, however as is usually the case in computer science, this means that more of the hard work must be done by the programmer.

**All of the above** The most ambitions possibility would be to emulate Java's Swing library, and simply provide all three methods of laying out a data structure, and let the user choose which is appropriate for them. One of my main focuses next semester will be attempting to

discover the minimum amount of meta data required to discern the structure of data, and use that information to provide the simplest interface possible to the display function.

## **5.2 Input Parsing**

The current input parsing system is somewhat inconsistent and could be improved by making it more like a compiler's parser, breaking items into tokens and evaluating each token. This would allow nesting statements and would allow the command prompt to be more realistic, and more accurately simulate programming line by line.

## **5.3 Time Control**

Currently there is no concept of time within the program. The data structure is in one state, and the user enters a command which moves the data structure to another state. I would like to add the ability to view more complex operations, such as the balancing of a binary search tree as multiple frames for one operation and allow the user to step back and forth between frames. This would allow the user to view a complex operation multiple times and more fully understand the operation.

## **5.4 Algorithm Animation**

I would also like to be able to show real code being executed and its effect on a data structure or set of variables. Showing for example, a program performing bubble sort on an array, or a program operating on a stack. This would allow this Visualization program to be used for any number of other applications. For example, demonstrating scheduling algorithms and the data structures used to store processes waiting to be scheduling.

## **5.5 Effectiveness Study and Curriculum Integration**

I plan to conduct an informal study of the effectiveness of the current implementation for teaching stacks and queues to a class of students who are learning them for the first time. I will be working with the professor the this class as a Teachers Assistant, and working with the professor to find ways to effectively include this tool, or others like it in the curriculum. I will also be informally surveying students in the class about how this tool did or did not help them learn about the data structure and how it could be improved to make it more effective for future uses.

## References

- [1] Baecker, Ronald. "Sorting out Sorting: A Case Study of Software Visualization for Teaching Computer Science", Software Visualization: Programming as a Multimedia Experience, MIT Press, 1998
- [2] Boroni, Christopher and Goosey, Frances and Grinder, Michael and Ross, Rockford. "A Paradigm Shift! The Internet, the Web, Browsers, Java, and the Future of Computer Science Education", ACM Press, 1998.
- [3] Brown, Marc and Sedgewick, Robert. "A System for Algorithm Animation", ACM Press, 1984
- [4] Johnson, W. Lewis and Rickel, Jeff, "Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments", International Journal of Artificial Intelligence in Education, 2000: 47-48.
- [5] Napps, Thomas et al. "Exploring the Role of Visualization and Engagement in Computer Science Education, Report of the Working Group on 'Improving the Educational Impact of Algorithm Visualization'", ACM Press, 2002.
- [6] Shaffer, Clifford A. and Heath, Lenwood S. and Yang, Jun. "Using the Swan data structure visualization system for computer science education", ACM Press, 1996.
- [7] Stasko, John and Badre, Albert and Lewis, Clayton. "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis", ACM Press, 1993
- [8] Wiggins, Melissa. "An Overview of Program Visualization Tools and Systems", ACM Online Portal [<http://portal.acm.org>]