

# A TOOL FOR DATA STRUCTURE VISUALIZATION AND USER-DEFINED ALGORITHM ANIMATION

Tao Chen<sup>1</sup>, Tarek Sobh<sup>2</sup>

**Abstract** -- In this paper, a software application that features the visualization of commonly used data structures and their associated insertion and deletion operations is introduced. In addition, this software can be used to animate user-defined algorithms.

**Index Terms** – Algorithm Animation, Data Structure Visualization, JavaMy.

## INTRODUCTION

Data Structures and Algorithms is a fundamental course in Computer Science. However, many students find it difficult because it requires abstract thinking. It would be very helpful if there was a visualization tool of data structures such as arrays, queues, stacks, trees and graphs for students to manipulate. The tool would allow students to see how an element is inserted into or deleted from different data structures, how a tree is traversed in different order (pre-order, in-order, post-order, level-order), etc. Moreover, this tool would provide a simple language, by which students can write their own algorithms so that the execution of the algorithm is animated. This project is intended to create an exploration environment, in which students can learn through experimentation. This tool can be used as an effective supplement to the traditional classroom education and textbook for Data Structures and Algorithms courses. The software package presented in this paper has the following functionality.

- Provides complete visualization for the widely used data structures such as array, stack, queue, tree, heap, graph, etc.
- Provides the animation of common operations associated with the data structures, such as inserting an element into and deleting an element from array, stack, and queue
- Provides animation of simple user-defined algorithms.

## BACKGROUND

The development of technologies and the evolvement of the World Wide Web have influenced education. Instructional Web sites and courses on the Web have grown dramatically. Web-based courses that consist of the syllabus, assignments and lecture notes are now widely used. Instructional Web

sites that are dedicated to Data Structures and algorithms can be easily found by using Search Engines. To name a few:

[http://swww.ee.uwa.edu.au/~plsd210/ds/ds\\_ToC.html](http://swww.ee.uwa.edu.au/~plsd210/ds/ds_ToC.html) [1]

<http://www.cce.hw.ac.uk/~alison/ds98/ds98.html> [2]

<http://www.cs.twsu.edu/~bjowens/cs300/> [3]

<http://www.cs.berkeley.edu/~edith/cs270/> [4]

However, The majority of the instructional web sites explored during this project lack interactive multimedia.

One of the best sites found that does contain interactivity is a course site developed for teaching Data Structures and Algorithms in Java by the Computer Science Department of Brown University [5]. This site has a collection of applets that demonstrate some commonly used data structures such as queues, stacks, and some famous algorithms such as merge sort, quick sort, etc. However, these applets are not complete and lack a common Graphical User Interface. Another good site in interactive Data Structure visualizations is developed by Duane J. Jarc in George Washington University [6]. This site provides animations in binary Trees, graphs, and sorting algorithms. But there is no animation available for algorithms that are defined by users.

Algorithm animation is a type of program visualization that is mainly concerned with displaying the executions of computer algorithms. Lots of work has already been done in this field. For example, the XTANGO [7] and POLKA [8] systems developed by the Graphic, Visualization and Usability Center (GUV) at Georgia Tech are general-purpose animation systems, which require the user to write an algorithm in the C language and register the events that the user wants to observe during the execution of the algorithm. However, these systems are implemented on top of Unix and X11 Window system, and are not portable to other platforms. In addition, we feel they are overkill for a basic Data structures and Algorithms course.

Another algorithm animation system found is Zeus[9], which is developed by Digital Equipment Corporation's Systems Research Center. This system is a little complicated, requires from the user lots of effort to prepare an animation. It is targeted at more advanced application programmers.

Since our software is intended to aid first year Computer Science students learning Data Structures and Algorithms, ease of use becomes our main consideration. Our approach for the user-defined algorithm animation is that the user codes the algorithm in a simple language called JavaMy, which is very similar to Java. The only effort the

<sup>1</sup> Tao Chen, University of Bridgeport, Department of Computer Science and Engineering, Bridgeport, CT 06601, taochen@bridgeport.edu

<sup>2</sup> Tarek Sobh, University of Bridgeport, Department of Computer Science and Engineering, Bridgeport, CT 06601, sobh@bridgeport.edu

user needs to make is to instantiate the data structures he/she wants to observe using the observable data types provided by the software. After parsing the JavaMy algorithm file, an animation frame is created and the observable data structures are added to the frame so that the user can watch the changes made to the data structures when the algorithm is executing.

### SOFTWARE PACKAGE

Before discussing the design of the software package, an overview of the functionality of the package is given here. The screenshots on the following pages should give an idea of how the software runs.

#### Data Structure Visualization

The observable data structures currently available in this software packages include: array, stack, queue, binary search tree, heap and graph.

- Array  
An Array stores a collection of identically typed objects, which are randomly accessible by a numeric index. The structure is shown in Figure 1.

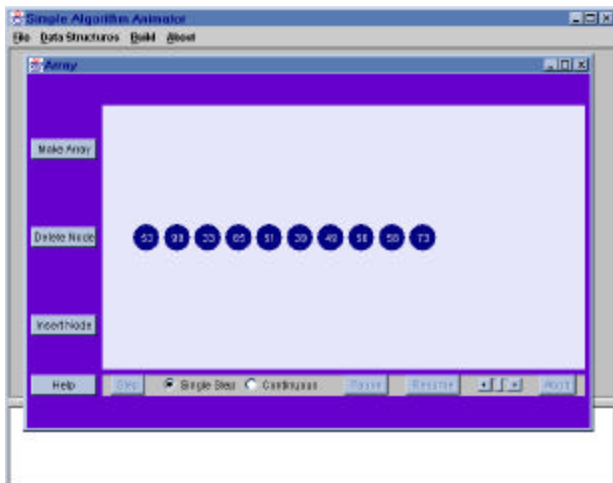


FIGURE 1  
ARRAY

- Stack  
A stack is a data structure in which all access is restricted to the most recently inserted element. The stack structure is shown in Figure 2.
- Queue  
A queue is a data structure that restricts the access to the least recently inserted item. The basic operations supported by queues are enqueue and dequeue, representing insertion to the rear (back) and removal of the item at the front. Figure 3 demonstrate the array-based queue structure.

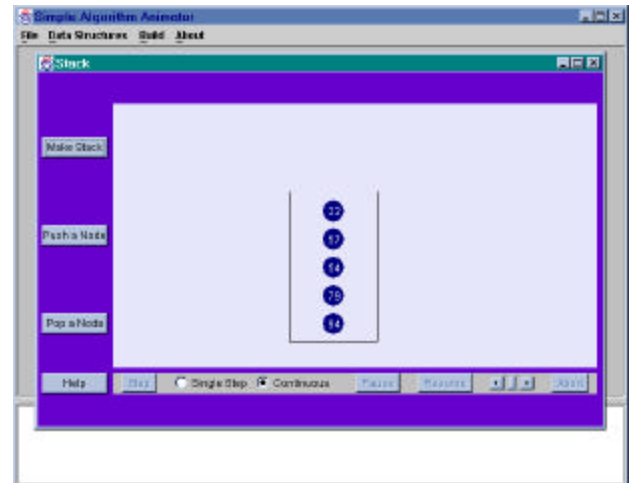


FIGURE 2  
STACK

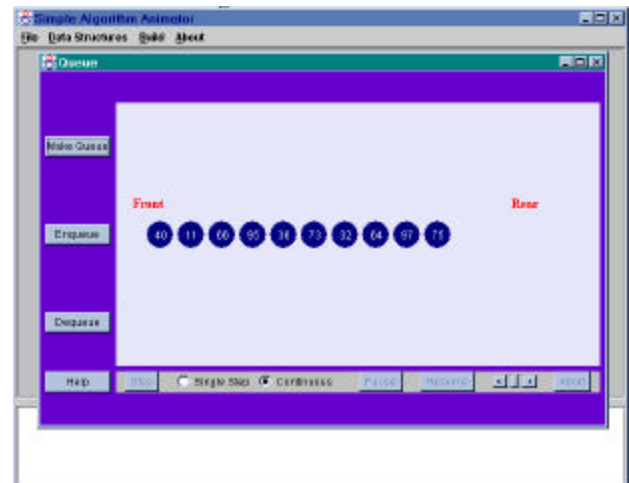


FIGURE 3  
ARRAY-BASED QUEUE

- Binary Search Tree  
A binary search tree is a kind of binary tree where every node's left subtree has values less than the node's value, and every right subtree has greater values. The basic operations are delete, insert and find. The structure is shown in Figure 4.
- Binary Heap  
A binary heap is a complete tree where every node has a key more extreme (greater or less) than or equal to the key of its parent. In our project, a Max Heap is implemented. The allowed operations are deleteMax and insert. The structure is shown in Figure 5.

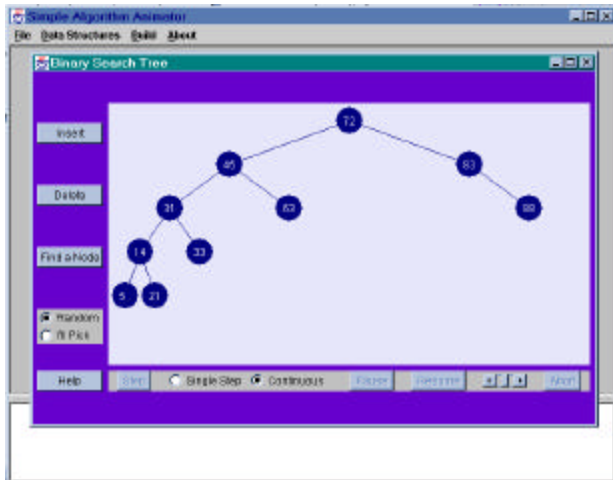


FIGURE 4  
BINARY SEARCH TREE

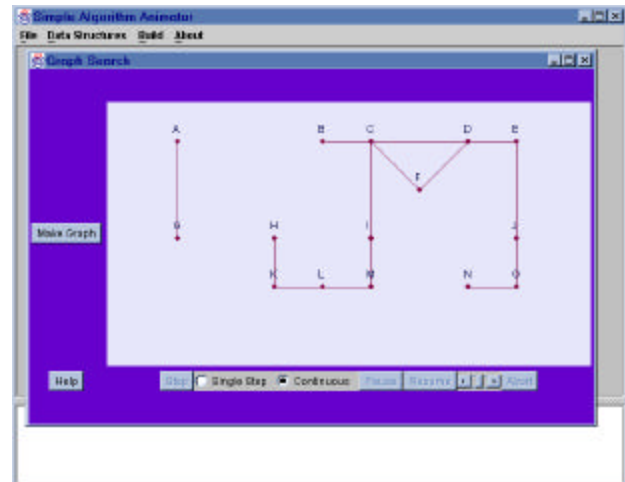


FIGURE 6  
UNDIRECTED GRAPH

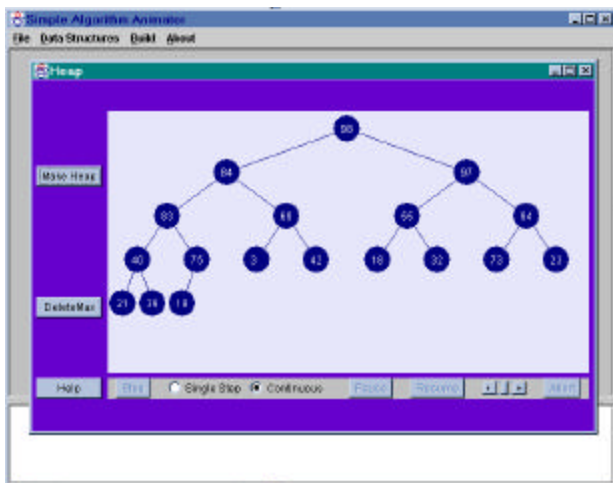


FIGURE 5  
HEAP

- Graph  
A graph consists of a set of vertices and a set of edges that connect the vertices as shown in Figure 6.

### User-defined Algorithm Animation

In this section we describe the steps of animating an algorithm and the details of the JavaMy language which is provided by the software and is used for visualizing the execution of the algorithm.

#### 3.2.1 Steps

To illustrate how the animation of a user-defined algorithm is done we will use a simple sorting algorithm -- bubble sort -- as an example to walk through the steps.

Bubble sort works by repeatedly moving the largest element to the highest index position of the array. The algorithm can be summarized as following.

1. Step through the array of data.
2. While stepping through, if two adjacent values are not in sorted order, then swap them.
3. When a complete pass of the data has been conducted, if any swaps have been made, then the data may still not be sorted. Goto 1.
3. Otherwise, if no swaps were made on the last pass, then the data is sorted, and the algorithm is finished.

Before we use the software to visualize the execution of bubble sort, we need to translate the above algorithm into JavaMy code. The details of the JavaMy language can be found in next sub-section. The bubble sort algorithm translated into JavaMy reads:

```
public static void main(String args[]) {
    final int SIZE = 8;
    MyArray intArray = new MyArray(
        AnimatorFrame.ARRAY_POSITION,SIZE);
    for (int i=0; i<SIZE; i++) {
        intArray.setValue(
            ScreenPanel.getRandom(10, 100), i);
    }
    for (int i=SIZE; i>1; i--) {
        for (int j=0; j<i-1; j++) {
            if (intArray.getInt(j) > intArray.getInt(j+1))
                intArray.swap(j, j+1);
        }
    }
}
```

The first step is to write the algorithm. The user can use any text editor to enter and edit the algorithm. To make it easy to use, the software provides a simple built-in code editor. By clicking File->New menu item, the user can enter the code into the text field.

The second step is to save the entered algorithm into a file by clicking File->Save as menu item as shown in Figure 7.

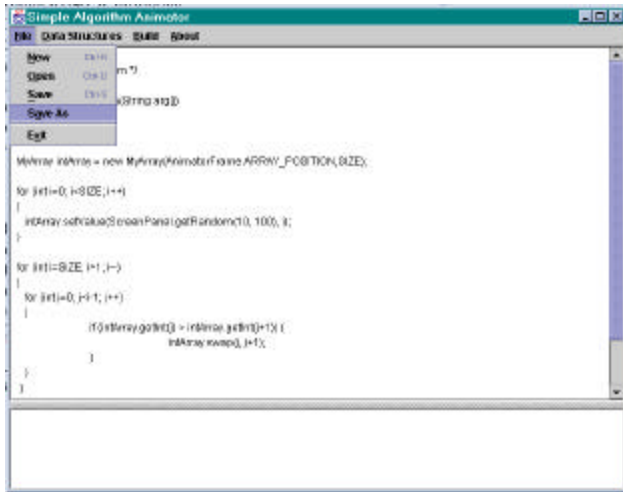


FIGURE 8  
SAVE ALGOIRTHM FILE

After saving the file, the user can parse it by clicking Build->Compile menu item as in Figure 9.

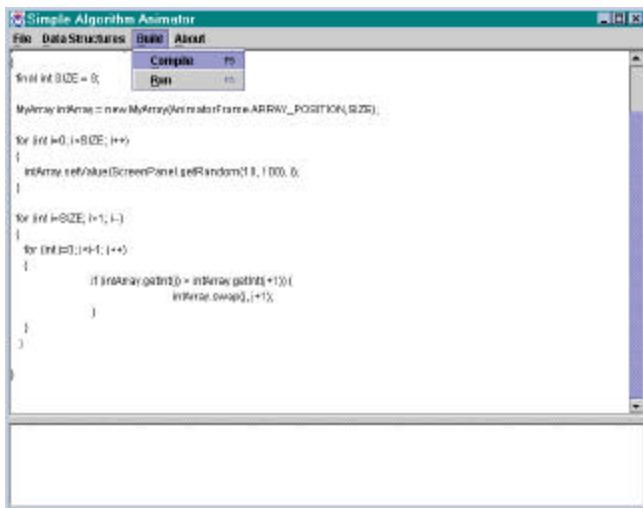


FIGURE 9  
PARSE AND COMPILE ALGORITHM FILE

If the file is edited using another editor, the user can load the algorithm file by first clicking the File -> Open menu item, then finding the file name in the File Open dialogue.

If no error occurs during the parsing process, the resulting Java file will be compiled. If errors occur during the parsing or compilation process, the errors will be displayed on the text area on the bottom of the window. The user can then go back to the algorithm file and make necessary corrections. If the file is parsed and compiled successfully, a corresponding message will be displayed as shown in Figure 10.



FIGURE 10  
PARSED AND COMPILED SUCCESSFULLY

Then the user can click Build->Run menu item to watch the animation. The animation frame is shown in Figure 11.

The animation frame consists of the animation canvas, the user algorithm text field, and the animation control panel. The animation canvas is where the data structures used in the algorithm are displayed. The user-defined algorithm coded in JavaMy language is displayed in the right hand text field. The control panel can be used to control the animation. The user can choose to run the algorithm animation either continuously or step by step by clicking the radio button labeled “Continuous” or “Single Step”. The animation speed can be changed by clicking the slider bar.

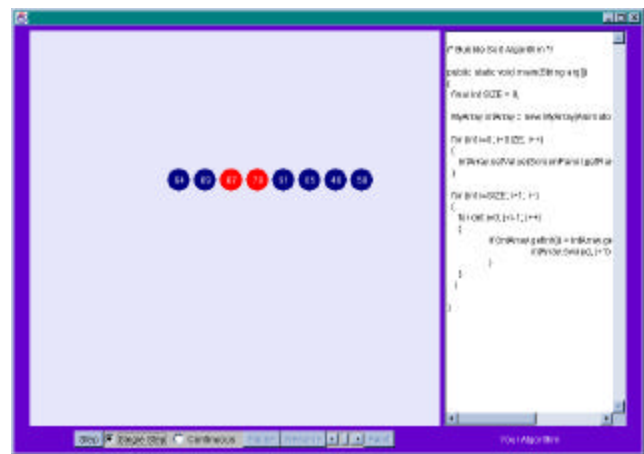


FIGURE 11  
ANIMATION FRAME OF BUBBLE SORT

### 3.2.2 JavaMy and Algorithm Coding

As mentioned earlier, JavaMy is the language used to code the user-defined algorithm. The syntax of JavaMy is similar to Java. The difference is in the program constructs. Every program in Java consists of at least one class definition. When the class definition is saved in a file, the file name

must be the class name followed by the “.java” file name extension. However, in JavaMy the user does not need to define a class, the coded algorithm is put into the main() method. That is, the user algorithm file always starts with

```
public static void main(String args[])
```

Moreover, the algorithm file can be named in anyway the user wants. However, the file name extension “.javamy” is recommended to separate the algorithm file coded in JavaMy from other files.

When coding the algorithm, the user is allowed to make a decision regarding which of the data structures used in the algorithm he/she wants to observe, and use the set of the observable data types provided by the software to define these data structures. All the observable data types are named by adding the prefix “My” to the corresponding normal data types. For instance, the observable array is named MyArray, and the queue is named MyQueue, etc. The data structure objects that do not interest the user can be instantiated by the data types provided by Java API. The software also provides some helper Java classes such as DrawableString, which can be used to add labels, explanations and other useful information to the Animation Frame. All the available observable data type classes, helper classes and their usage can be found in the Javadoc documentation that comes with the software.

In the following, a balanced symbol checking algorithm will be used as an example to demonstrate how the algorithm is coded in JavaMy language and what the final animation looks like.

A balanced symbol checker is a tool to help debug compiler errors, which checks whether symbols are balanced. In other words, whether every “{” corresponds to a “}”, every “[” to a “]”, every “(” to a “)”, and so on. The basic algorithm is stated as follows:

Make an empty stack. Read tokens until the end of the input file. If the token is an opening symbol, push it onto the stack. If it is a closing symbol and if the stack is empty, report an error. Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, then report an error. At the end of the file, if the stack is not empty, then report an error.

The above algorithm coded in JavaMy is shown in the following program:

```
/* Balanced Symbol checker is used to check whether every
   { corresponds to a }, every [ to a ], every ( to a ). And the
   squence [()] is legal, but [(]) is wrong. */
```

```
public static void main(String arg[]) {
    String input = "{[(())}";
    char c, match;
    String errmsg;
    MyArray in = new MyArray(
        AnimatorFrame.ARRAY_POSITION,
        input.length());
    MyStack pendingTokens = new MyStack(
        AnimatorFrame.STACK_POSITION, 0);
    for (int i=0; i<input.length(); i++) {
```

```
        in.setValue(input.charAt(i),i);
    }
    for (int i=0; i<input.length(); i++) {
        c = in.getChar(i);
        switch(c) {
            case '(': case '{': case '[':
                pendingTokens.push(c);
                break;
            case ')': case '}': case ']':
                if (pendingTokens.isEmpty()){
                    System.out.println("Extraneous "
                        + c + " found");
                }
                else {
                    match = pendingTokens.topChar();
                    pendingTokens.pop();
                    if (match == '(' && c != ')' ||
                        match == '{' && c != '}' ||
                        match == '[' && c != ']') {
                        errmsg = "Found \"" + c + "\"
                            does not match \""+match+"\"";
                        JOptionPane.showMessageDialog(
                            new JPanel() , errmsg, "Error",
                                JOptionPane.ERROR_MESSAGE);
                    }
                }
                break;
            default:
                break;
        }
    }
    while (!pendingTokens.isEmpty()) {
        match = pendingTokens.topChar();
        pendingTokens.pop();
        errmsg = "Unmatched \"" + match + "\"";
        JOptionPane.showMessageDialog( new
            JPanel(), errmsg, "Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

The program starts with multiple-line comments. The comment notation in JavaMy is the same as Java. Multiple-line comments are delimited with /\* and \*/, and single-line comments are delimited with //. Following comments is simply a blank line. Blank lines, space characters and tab characters are known as white-space. Such characters are used to make the program easier to read. They are ignored by the parser. The bold line indicates the beginning of the real code of the algorithm. The three lines following the opening parentheses declare normal variables as in Java, using the data type provided by the Java programming language. The next five bold lines instantiated two observable data structures that will show on the animation frame. The first one is an array, which is used to hold the input string, that is, the string to be checked. The second one is a stack, which is used to hold the opening

symbols. Here, MyArray and MyStack are used. Both of the constructors of MyArray and MyStack take two parameters. One is the Position parameter, which is used to decide the location of the data structure on the animation frame. Another parameter is the size of the array or stack. The rest of the code is the same as Java. Class MyArray and MyStack provides most of the commonly used methods, for example, setters and getters for setting and getting the values of the elements in the array, respectively, push(), pop() and methods for peeking the top element on the stack, etc. Details of those methods are described in the documentation generated by Javadoc.

After parsing and compiling the algorithm successfully, we can run the animation as described in subsection 3.2.1. The resulting animation frame is shown in Figure 12.

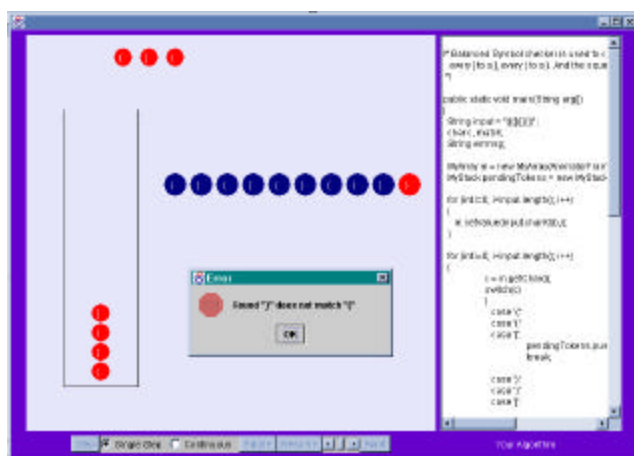


FIGURE 12

ANIMATION FRAME OF BALANCED SYMBOL CHECKING

## Implementation

This software package is implemented using Java. Java is a general-purpose object-oriented language. The AWT and Swing packages of Java provide extensive components for creating Graphic User Interfaces. Moreover, its graphics capabilities are platform independent and hence portable, which makes it our natural choice for implementation.

To animate a user-defined algorithm, a lexical analyzer and parser are needed. A lexical analyzer breaks an input stream of characters into tokens. A parser reads the input tokens and converts the tokens to a Java program. There are several ways to build a lexer and parser. One possibility would be to code the lexical analyzer and parser completely from scratch, implementing all string handling and checking functions, which is a very tedious and error prone process. Another method is to find a Java parser generator, which reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. After intensive search, we found that JavaCC [10], a product of Sun Microsystems is currently the most popular parser generator for use with Java applications.

0-7803-6669-7/01/\$10.00 © 2001 IEEE

Consequently, it was our choice. The parser is generated by two steps: (1) Run JavaCC on the grammar input file to generate a set of Java files that implement the parser and the lexer. (2) Compile all the Java files obtained in step (1).

## Conclusions and Future Works

In this paper, we present a visualization tool designed to aid first-year computer science students learn Data Structures and Algorithms. This tool not only lets students visualize the commonly used data structures, but also allows students to write their own algorithms in a Java similar language - JavaMy, and observe the execution of the algorithms. We believe this tool will be an effective supplement to traditional instruction.

Because of the time limitation, only the most commonly used data structures are implemented in this version of the software package, which include arrays, stacks, queues, binary search tree, binary heap, priority queue and undirected graph. There are two ways to add more observable data structures to this software such as directed graph, weighted graph, AVL tree, Red Black Tree, AA-tree, splay tree, hash table, etc. One way is to implement these data structures in the software. Another approach would be to develop and implement a mechanism for the software package to recognize the user-defined observable data structures, and leave the implementation to the user. This approach will allow users to use their own observable data structures, hence add more flexibility to the software.

Another possible future enhancement for the software is to highlight the executing command line of the user-defined algorithm file. This would help the user to better follow the execution of the algorithm.

## References

- [1] Morris, John, "Programming Languages and Data Structures", [http://swww.ee.uwa.edu.au/~plsd210/ds/ds\\_ToC.html](http://swww.ee.uwa.edu.au/~plsd210/ds/ds_ToC.html)
- [2] Cawsey, Alison, "Data Structures and Algorithms", <http://www.cce.hw.ac.uk/~alison/ds98/ds98.html>
- [3] Owens, Brad "CS300 Data Structures and Algorithms I", <http://www.cs.twsu.edu/~bjowens/cs300/>
- [4] Cohen, Edith "CS270: Combinatorial Algorithms and Data Structures", <http://www.cs.berkeley.edu/~edith/cs270/>
- [5] Goodrich, Michael T. and Tamassia, Roberto, "Data Structures and Algorithms in Java", <http://www.cs.brown.edu/courses/cs016/book/>
- [6] Jarc, Duane J., "Interactive Data Structure Visualizations", <http://www.seas.gwu.edu/~idsv/idsv.html>
- [7] The Graphics, Visualization & Usability (GVU) Center at Georgia Tech, "XTango", <http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html>
- [8] The Graphics, Visualization & Usability (GVU) Center at Georgia Tech, "Polka", <http://www.cc.gatech.edu/gvu/softviz/algoanim/xtango.html>
- [9] System Research Centers (SRC) at Compaq Computer Corporation, "Algorithm Animation at SRC", <http://www.research.compaq.com/SRC/zeus/home.html>
- [10] Sun Microsystems, "JavaCC - The Java Parser Generator", <http://www.metamata.com/javacc/>

October 10 - 13, 2001 Reno, NV