MULTIMEDIA VISUALIZATION OF ABSTRACT DATA TYPE

By

CONG XU

Bachelor of Science

Ocean University of Qingdao

Qingdao, China

1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1997

OKLAHOMA STATE UNIVERSITY

MULTIMEDIA VISUALIZATION OF

ABSTRACT DATA TYPE

Thesis Approved:

Jacques E. LaFrance

Thesis Adviser

J Chandler

Wayne B Powell

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation and respect to my graduate advisor, Dr. Jacques LaFrance, for his intelligent supervision, constructive guidance and friendship. My sincere appreciation and respect extend to Dr. J. P. Chandler and Dr. K. M. George for serving on my committee. Their advice and support are invaluable throughout my study. Without their encouragement and patience, this thesis would have been impossible.

Moreover, I wish to express my sincere gratitude to Computer Science Department, Oklahoma State University for supporting during my study.

My in depth gratitude and respect go to my parents Mr. Jiashu Xu and Ms. Junrui Sun, for their love, encouragement, support and confidence on me. Without this, I wouldn't have been what I am today. Last but certainly not least, I thank my brother, Mr. Xiaoman Xu, and all other family members for their support and understanding.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figures Page

CHAPTER I

INTRODUCTION

In order to understand the complexity of the real world, people build abstract descriptions or models. By omitting details and including just the relevant factors, a model can provide a useful tool for understanding a particular problem. A model must be proved, tested and solved. Systematized techniques for analyzing models are called algorithms [22]. Models and algorithms must be represented in a way that is clear and informative.

"A picture is worth a thousand words." Visualization is a method of computing [33]. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Today, the study of visualization has expanded greatly with the variety of representations [33], such as computer multimedia, graphics, and graphical user interfaces. Visualization uses carefully designed representations to help people understand complicated models and algorithms. Nowadays visualization has become one of the most exciting and rapidly growing fields in computer science. It has greatly affected such diverse areas as education, science, industry, business, and military than ever before. One of the benefits of using visualization in education is that it involves the right side of the human brain in the learning activity [21]. The perceptual work is

done by the right hemisphere of human brain that initially processes pictorial information. Then the left hemisphere does the analysis work. If we use both graphics and words, more of the brain will be involved in processing the information. This results in better and fast understanding of abstract concepts.

Anyone who has written or studied computer programs has encountered data structures since every program has one or more data structures. An Abstract Data Type (ADT) is a mathematical abstraction with data and a set of operations defined on that data [1]. A data structure consists of data and the operations that can be applied to that data. The use of abstract data types for data structures is very important for the development of modular programs.

The study of data structures would be much easier if the student were able to visualize the graphical representation of the structure and test various operations related to the data structure. Knowing of these benefits that graphics can provide encourages us to explore visualization aided tools for learning data structures. This study focuses on the simulation of three of the basic data structures in computer science: lists, stacks and queues. Virtually every significant program will use at least one of these structures explicitly, and a runtime stack is always implicitly used in the program [46].

The list ADT has the form $a_1, a_2, a_3, \ldots\ldots a_n$ with the size n, and a_{i+1} follows a_i ($i<n$). Associated with this data, some operations are insert, delete and find. The stack ADT is a list with the restriction that inserts and deletes can be performed at the end of the list. Related operations are push and pop. The queue ADT is a list where insertion is done at one end, and deletion is performed at the other end. The basic operations are enqueue and dequeue.

The primary objective of this thesis is to build a simulator to help the user execute and visualize the immediate effects of each step of an operation on the three ADTs, as well as explain the changes in visualization using sound. The emerging representation formats such as sound [3] [7] [13], and animation [44] are much less developed than those for static graphics. I will explore both sound and animation representation formats.

My thesis is to design and implement a flexible and user-friendly program for simulating data structures stated above. We called it "VisualADT 1.0" system. This visualization system is built and run on the Microsoft Windows 95 operating system. This means that the application is an event-driven program. This software will be developed using Macromedia Director 5 PC version and will be a 32-bit application that runs under Windows 95, Windows NT authoring environments as well as MacOS on both 68K and PowerPC systems. Developers also can playback the movie on OS/2, SGI, OS/9, many interactive TV formats, and the Internet with Shockwave plug-in for Director [32]. This provides a smooth and flexible open system user interface at the time that users are accustomed to window-style interfaces.

The remaining chapters of this thesis are arranged as follows: Chapter II provides some background information in the areas of data structure, visualization and multimedia. Chapter III describes related works and Chapter IV discusses software design decisions and implementation issues. Finally, Chapter V summarizes the thesis and suggests the future research possibility.

CHAPTER II

LITERATURE REVIEW

2.1. Data Structure Fundamentals

The computer has become an indispensable tool of scientific research, business and industry during the 1960s. The computer is an automaton that executes computational processes according to precisely specified rules [47]. The essence of the computer's power lies in its ability to execute long sequences of instructions which contain an almost infinite combination of elementary actions. Programming is the act of incorporating such instruction sequences into "recipes" representing certain classes of computational processes [47]. Computer programming has advanced from a craft to an academic discipline in the early 1970s. The initial outstanding contributions toward this development were made by E. W. Dijkstra and C. A. R. Hoare [36]. *Structured Programming* [14] opened a new view of programming as a scientific subject. Programs are concrete formulations of abstract algorithms based on particular representations and structures of data. Hoare brought order into the bewildering variety of terminology and concepts on data structures. Wirth [47] introduced programming as the art or technique of constructing and formulating algorithms in a systematic manner.

In particular, data structures were commonly modeled by graphs because of

their generality [18]. However, such schemes didn't bundle a data structure and its operations into a package. The language Simula was the first to provide facilities for constructing such packages of data types and associated operations, calling them classes [15]. Some languages that provide facilities for defining and using protected ADT's are Alphard via forms [38], CLU via clusters [27], and Mesa and Modula via modules [20][48]. The methods of specifying data abstraction in these languages are explicit. The semantics of the new data type are modeled constructively in terms of operations upon more basic data types [41].

ADT is a set of objects with a set of operations. The structure of an ADT encourages modular programming. We can build new ADTs by writing new operations and by using the object structuring operators such as union, list and struct. Each program will use at least one data structure. Lists, stacks and queues are three fundamental ADTs in the family of data structures. These data structures are linear structures which means that there is a natural linear ordering among the elements of the structure. There is another linear structure: deque (double-ended queue, pronounced "deck"). The distinguishing characteristic of deques is that insertions and deletions are made at either end and only at the ends. The major operation on deques are insertfront, insertback, removefront and removeback. We can consider that queues and stacks are special deques [49].

2.1.1 The List ADT

A linear list is a sequence in which one can insert, delete or access items at any point [49]. The general list has the form $a_1, a_2, a_3, \ldots\ldots, a_n$ with the size n. A special

list with size 0 is a null list. Except the null list and the list of one element, the item a_{i+1} succeeds a_i ($i< n$) and a_{i-1} precedes a_i ($i>1$) in any list. The elements in the list can be integers or arbitrarily complex elements. To simplify matters, we assume that the elements are characters. There are a set of operations that perform on the list ADT. Insert and delete are the two important operations of the list, which inserts and deletes the specified key from the list. Some operations are print_list and make_null, which do the obvious things; find, which returns the position of the first occurrence of a key. We also can add some operations such as next and previous, which would take a position as a argument and return the position of the successor and predecessor, respectively. There is no rule telling us which operations must be supported for each ADT; this is a design decision and up to the programmer.

All of the instructions on the list can be implemented by using an array. This is limited by over-estimating the memory space and running time. Another way is using a linked list. Figure 1 shows the general idea of a linked list. The linked list consists of a series of structures. Each structure contains the element and a pointer to a structure containing its successor, which is called the next pointer. The linked list is not stored contiguously and it provides fast running time for insertions and deletions.

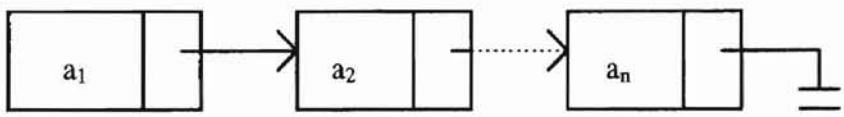


Figure 1: A Linked List

Sometimes it is convenient to traverse lists backwards. By adding an extra field to the data structure, containing a previous pointer, we get a doubly linked list. Another convention is the circularly linked list, by making the last cell keep a pointer back to the first. By joining the two, we can get the double circularly linked list.

2.1.2 The Stack ADT

A stack is a last-in-first-out list, or LIFO data structure. Figure 2 shows a stack model. A stack is a list with the restriction that inserts and deletes can be performed on the top of the stack. The fundamental operations on a stack are push and pop. Push is equivalent to an insert and pop deletes the latest inserted element, i.e. the top element. Stacks only can access the top element. Stacks can use pointers or arrays to be implemented.

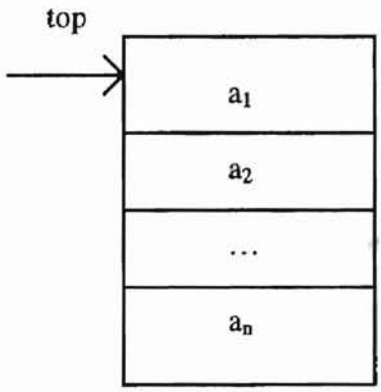


Figure 2: A Stack

2.1.3 The Queue ADT

Queues are first-in-first-out lists, or FIFO. The difference is that insertion is done at one end and deletion is done at the other end. The basic operations on a queue are

enqueue and dequeue. Enqueue inserts an element in the rear of the list and dequeue deletes the element at the front of the list. Figure 3 shows the abstract model of a queue. Queues can be implemented by the linked list and array.

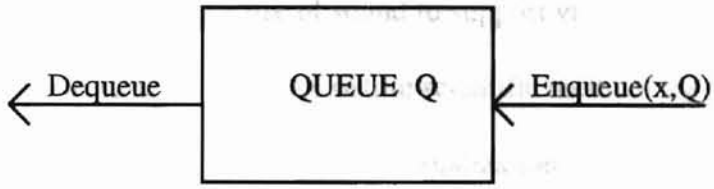


Figure 3: A Queue

2.2 Visualization Fundamentals

Program visualization systems can be classified by whether they illustrate code or data structure, and whether displays are dynamic or static [10]. In addition, dynamic displays are either interactive such as computer games or passive such as a videotape. Flowcharts are typical static displays of program codes and logic. Static displays of program data structure are more difficult than static displays of code since there are many different ways to implement a certain data structure. Dynamic displays not only have all the problems that static displays have but also they have more problems such as how to update and show the changes in the display. How to create the effective display involves many complex factors, particularly including human's vision. People remember 20% of what they see, 40% of what they see and hear, and 70% of what they see, hear and do. This is also the basis for the learn-by-doing philosophy embedded in my project.

Graphics, text, and animation exploit only the human vision system. For presenting complex information, to exploit more sense will be helpful. Except video games, most computer graphic applications are silent. With the proliferation of more capable sound hardware and software, it is now possible to use more and better sound in visualization. Some call the use of sound to support visualization sonification [22]. The goal of visualization is to affect existing scientific methods by providing new scientific insight through visual methods. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science. In computing, scientists are using graphics and animation to help visualize data. McCormick [33] claims that the magnitude of scientific breakthrough with visualization techniques is comparable to that realized from the introduction of the supercomputer to scientific computing. We can only imagine how much more information will be communicated and assimilated when interactivity and sound are added to scientific visualization [6]. Visualization is a tool both for interpreting image data fed into a computer, and for generating images for complex multi-dimensional data sets. Visualization of data structures will help the user comprehend the task of an algorithm and explore different scenes in the construction of the data structures. This is suitable in laboratory exercises and distance learning as a teaching and research tool.

2.3 Multimedia Fundamentals

In the past 40 years, the computer has evolved from a complex adding machine to a device that begins to reflect the mental and sensory abilities and demands of human

beings. The computer has become a ubiquitous component of modern life: computers compete with us in games, teach children and entertain us. The flexibility and low cost of the computer have enabled us to adapt the machine to our everyday needs.

However, computers lack the analogues of eyes, ears, voice and senses of touch. Now we have video cameras, microphones, speakers to increase the communication capability between computers and human. We realize that it is important to interact with computers using all of our senses and communication abilities. Bly [8] found that sound plus graphics was more effective than just graphics. Voice annotation is also now relatively easy, particularly since microphone input is becoming a common feature of personal computers. Some applications are impossible without the use of multimedia interfaces. Sometimes multimedia enhances what would be cumbersome task of input/output. For example, instead of typing commands, spoken commands can leave the hands free to operate another device.

A medium is a carrier of information. A multimedia computer system is one that is capable of the input or output of more that one medium [6]. Typically, the term is applied to computers that support more than one physical output medium, such as a computer display, audio and video. The goals of multimedia are to build better communication between human and computers. Multimedia systems strive to take the best advantage of human senses in order to facilitate communication. Communication between users and computers takes on four different forms: human-to-human (via computer), human-to-computer, computer-to-human, and computer-to-computer [6].

The study of multimedia user interfaces has not matured into an independent

discipline. So we must examine a range of research contributions in disparate areas that contribute to our understanding of these new interfaces.

CHAPTER III

RELATED WORKS

Visualization as a means to understand models and algorithms has been a subject of study since the 1960s. In the early days visualization of algorithms was done more in video tapes and films. This means that the visualization was static. The user cannot participate in the process of simulation. *L6: Bell Telephone Laboratories Low-Level Linked List Language*, the first computer-generated movie, was produced by Knowlton [24] in 1966. It showed how an assembly level list-processing language works.

There are three other important films: Hopgood's film on hashing algorithms (1974), Booth's (1975) *PQ-trees* [9] and Baecker's (1981) *Sorting Out Sorting* [2]. Hopgood's movie showed the actions according to different types of input using a hash table and a graph. The significant value of this movie was that it showed a data structure that was too large to have been computed by hand. Booth's film indicated several algorithms on PQ-trees. When the PQ-tree is changing, the movie showed a smooth transition. Baecker's *Sorting Out Sorting* illustrated a number of different sorting algorithms running on small and large data sets. When a change occurred in the data structure, it dynamically showed the changes. The movie also contained a sound track even though the sound did not provide more insight to the workings of the algorithms.

In the mid-1970s, a lot of work in algorithm animation systems was done at the University of Toronto under the direction of Ronald Baecker. Yarwood's master thesis *Toward program illustration* [50] and De Boer's *A system for animating micro-PL/I programs* [16] are among those systems built in 1974.

BALSA(Brown University Algorithm Simulator and Animator) algorithm animation system was used in the "Electronic Classroom" at Brown University in the 1980s. BALSA-I [10] was developed in the early 1980s and influenced by Smalltalk. It has good interactive environment and some Smalltalk techniques such as popup menus, overlapping windows and changing the cursor's shape to give detailed feedback. BALSA-II [10] was developed upon BALSA-I later which adopted a lot of the Macintosh user interface such as zoom in/out, dialog boxes and presenting both detailed and overall views of an object simultaneously. London and Duisberg [28] simulated a collection of algorithms by using Smalltalk's MVC(Model-View-Controller) paradigm and following BALSA's approach of annotating the algorithm in 1985.

Duisberg, in his Ph. D. thesis, investigated using temporal constraints to describe the appearance and structure of a picture as well as how those pictures evolve over time in the Animus System in 1986 [17]. Bentley and Kernighan [5] followed the BALSA paradigm implemented a set of tools for producing animation of algorithms in 1987. Kleyn and Gingrich [23] applied the BALSA animation techniques in their object-oriented system called GRAPHTRACE in 1988. PECAN program development system (1988), primarily a program development system, presents multiple views of the user's program and its semantics [37]. Early systems at Toronto, BALSA and the systems it influenced use a level of indirection which is extra and unnecessary baggage.

Researchers from computer science have focused on representations of data structures. A variety of commercial systems, subroutine libraries are available to create, store and draw graphs. Several systems have been built that automatically produce a static graphical display of a program's data structure. They do not show which operation is being performed and how the data structure changes. *Incense: A system for displaying data structure* [35], built by Brad A. Myers (1983), is a prototype that lets the user view the data structures in a desired fashion. The user can specify the variable name to get its graphical display. The user also can select one of those formats associated with each data type. The PV(Program Visualization, 1985) [11] project attempted to use graphics in all phases of programming large systems. Baskerville's (1985) GDBX [4] system also provided graphical displays of arbitrary data structures for debugging. GDBX ran on a Sun Microsystems workstation integrated with the standard UNIX debugger, DBX. PROVIDE [34] by Moher (1985) supported only direct views of data structures with non-user-defined variables. Lee (1988), in his M.S. thesis, designed a data structures display system which:

1. graphically displays a variety of data structures, including B-Tree, binary search tree and link-list.
2. allows the user to execute and study the immediate effects of each step of an operation on a particular data structure [26].

Lee's display system is implemented on VT100 type terminals which is rigid because it displays graphics as characters rather than as a combination of pixels. Stastko (1990) introduced the path-transition in TANGO animation systems [43]. TANGO is a framework that simplifies animation design. The path-transition paradigm is based on

four abstract data types: locations, images, paths, and transitions. A tool called DANCE (Demonstration ANimation CrEation) was also created. Shimomura's (1991) program: *Linked-List Visualization for Debugging* [40], which improved the visualization of linked lists under VIPS system. VIPS system uses UNIX's symbolic debugger DBX to execute the program to be debugged. Shimomura met the following requirements: easy shape recognition, easy change detection, selective display and rapid drawing. *Using Visualization Tools to Understand Concurrency* by Zernik et al (1992), used graphs to provide a logical view of execution [51]. Views are organized by computational messages, threads, and synchronization events. This tool provided a clear picture of concurrency by overcoming the concurrency bugs.

Arra's (1992) thesis, *Object-Oriented Data Structure Animation* [1], displayed the dynamic changes in the data structure and only considered dynamic program animation. The system contains a user-friendly Graphical User Interface(GUI) and a library that aids the animation of data structures. Knuth (1993) developed a set of standard data structures and a subroutine library, *the Stanford GraphBase* [25], to allow programmers to construct graph algorithms easily. The book is a collection of dataset and computer programs that generate and examine a wide variety of graphs and networks.

Shen's (1994) TBDSV system, allowed the user to choose from a set of tree-based algorithms and to explore the dynamic behavior of algorithms [39]. This system is built on the X windows system and simulated AVL tree, Red-Black tree, B-tree and splay tree. This system cannot accept user's input algorithms for visualization.

The most recent work in algorithm animation is done at the University of Southern California. Their project "Animating Algorithms, I" (1997), developed

interactive animation of several types of binary search trees using Java. The algorithms implemented so far include the following [45]:

1. Standard, garden-variety binary search trees.
2. Splay trees [Sleator and Tarjan], which are a self-adjusting data structure based on the splay operation. The applet (also shown here) uses bottom-up splaying. Another demo by Kindred and Sleator implements the top-down version.
3. Red-black trees, an implementation of 2-3-4 trees based on binary trees.
4. Randomized binary search trees based on treaps [Aragon and Seidel]. These provide an elegant BST-based alternative to Pugh's randomized skip lists.

The user can perform basic operations such as insertion, location and deletion on these data structures.

CHAPTER IV

DESIGN AND IMPLEMENTATION ISSUES

4.1 Implementation Platform and Environment

Macromedia Director 5 is the most powerful authoring tools for multimedia production and for the Internet. It was introduced by Macromedia Inc., the leader in digital arts, multimedia and Web publishing software, in March 1996. It runs on both the Mac and PC. Director 5 supports authoring on MacOS (68K and PowerPC), Windows 3.1, Windows95, and Windows NT systems, which creates true cross-platform multimedia products. The Lingo scripting language of Director 5 can precisely control text, sound, and digital video. Scripting languages are combinations of words that convey information and instructions. Lingo has certain commands and rules that you follow to create statements. Lingo also has some concepts which are very similar to object-oriented programming language. The following table tells you which Lingo terms correspond to which common object-oriented programming terms:

Lingo term	Object-Oriented term
Parent script	Class
Child object	Class instance
Property variable	Instance variable

Handler Method

Ancestor script Super class

Brown [12] noted that "Small amounts of data are good for introducing a new algorithm, whereas large amounts of data are good for developing an intuitive understanding of an algorithm's behavior." Director 5 has 48 sprite channels, which means we only can have 48 objects on the stage at the same time. I plan to animate the three ADTs of 10 nodes each. This is enough to illustrate the operations on these ADTs and introduce the new concepts. We also can insure that the growing data structure stays inside the border of the window.

The VisualADT software package allows the user to view and study an abstract data structure and its associate operations. The development process focuses on visualization design and user interface design.

4.2 Visualization Design

In the visualization design, we must first take the human visual perception into consideration. In order to enable the user to perceive the desired information efficiently, we divide the system into two components: static displays and dynamic displays. Static displays show the text of data, the image of data structure and the overall structures. Dynamic displays show the behavior of an operation and indicate the changes in the data structure.

My thesis also focuses on the use of color and sound, previously less explored areas in animation. Color has the potential to communicate lots of information efficiently. We use color for emphasizing patterns and highlighting activity. Director 5's paint tools

work the same as paint tools in applications such as Photoshop. The paint window offers a complete set of paint tools and inks to create cast members in the movie. The paint and cast windows share a dynamic link that anything you draw in the paint window becomes a cast member automatically and is displayed in the cast window.

We strongly concur with Gaver [19] that

> "Auditory displays have the potential to convey information that is difficult or awkward to display graphically. Sound can provide information about events that may not be visually attended, and about events that are obscured or difficult to visualize. Auditory information can be redundant with visual information, so that the strengths of each mode can be exploited. In addition, using sound can help reduce the visual clutter of current graphic interfaces by providing an alternative means for information presentation."

In my thesis the most obvious use of audio is that it reinforces visual views. For example, when the user clicks the list button on the stage, there is voice that says "You selected list data structure." When the user inserts or deletes a node from the data structure, there is voice message that says "You inserted or deleted a node." If you perform some illegal operations, such as you try to delete when there is no node in the list, there will be an alarm dialog box popped up with text message, a system beep and voice message that says "Sorry, the list is empty, please insert node."

4.3 User Interface Design

For a software as a teaching and learning tool, the importance of the user interface can not be overemphasize. Without the hardware limitations, the user interface has become more important and thus, we place more of the effort on software usability. During the design and development of the system, I consider the following principles [42]:

a. The interface should be appropriately consistent.

b. The interface should use terms and concepts which are familiar to the anticipated users.

c. The interface should include some mechanism which allows users to recover from their errors.

d. The interface should incorporate some form of user guidance.

e. The interface should respond with an appropriate amount of information.

f. The interface should provide an easy way out.

g. The interface should allow the user to work in real time.

h. The interface should minimize loading.

i. The interface should make states visible.

j. The interface should detect the user actions quickly.

The whole idea is to build an easily used and reliable system. The user can control the VisualADT system though the pull-down menus. There are two advantages of using menus [26]: firstly, explicit options are given, eliminating the possibility user typing mistakes; secondly, displayed options also serve as memory aid. We keep the menus short and include only the main operations on the three data structures. We have the

HELP menu which can provide instructions to the user on how to use the system. The user can directly interact with the system and the system responds in real time. The user is able to view the motions in a smooth way through the continuous transition. The importance of continuous transition [28] is: "Often it is even better to show smooth transitions between states; if a structure changes and the new state simply flashes on the screen, the viewer is typically startled and cannot see immediately without some mental effort how the new image evolved from the previous one." However, it is not an easy job to achieve the smooth transitions of action. With the powerful animation software, we get the most impressive smooth motion.

4.4 Project Design Decisions

Since I have started working on the data structure visualization project, I have made some decisions on designing the system. This project is very flexible in the following ways. Firstly, the user is able to select one of the three data structures (list, stack and queue), and to select any operations associated with them. Secondly, the user is able to input the test data at run-time. Thirdly, the user is able to watch the visualization as well as listen to the explanation.

At the beginning , one data structure must be created. Before any operation is executed, the user is able to select one of the three data structures to create it. The following are the decisions:

a. The user is able to select the options from the pull-down menu. After the "Welcome" window (Figure 4) moves out of the stage, the "Instruction" (Figure 5) window moves in. Sound accompanies the changing windows,

such as "Welcome to the VisualADT 1.0, the software for learning data structures stack, queue and list. Enjoy it!" and "Please read the following instructions." After the "Instruction" window moves out, there is voice message "Please select the data structure from 'File' menu 'New' option."

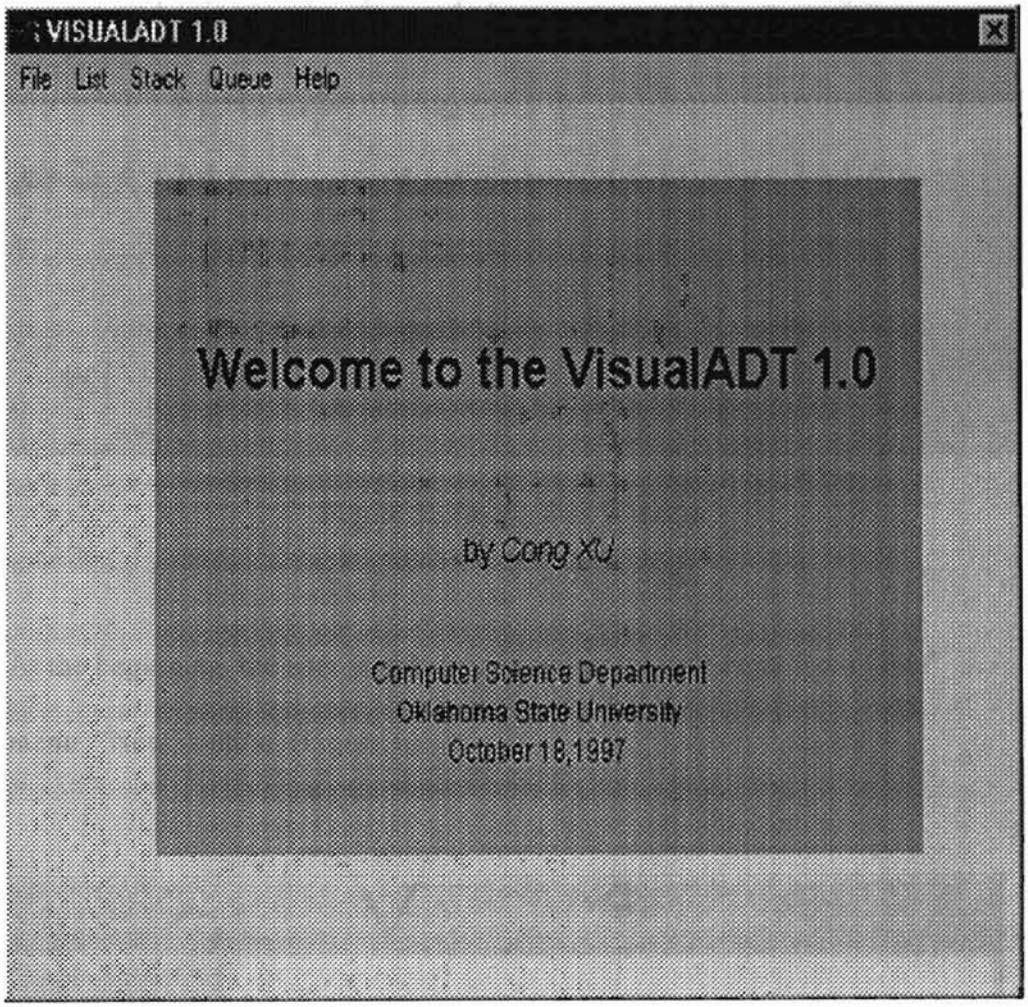


Figure 4: Welcome Window

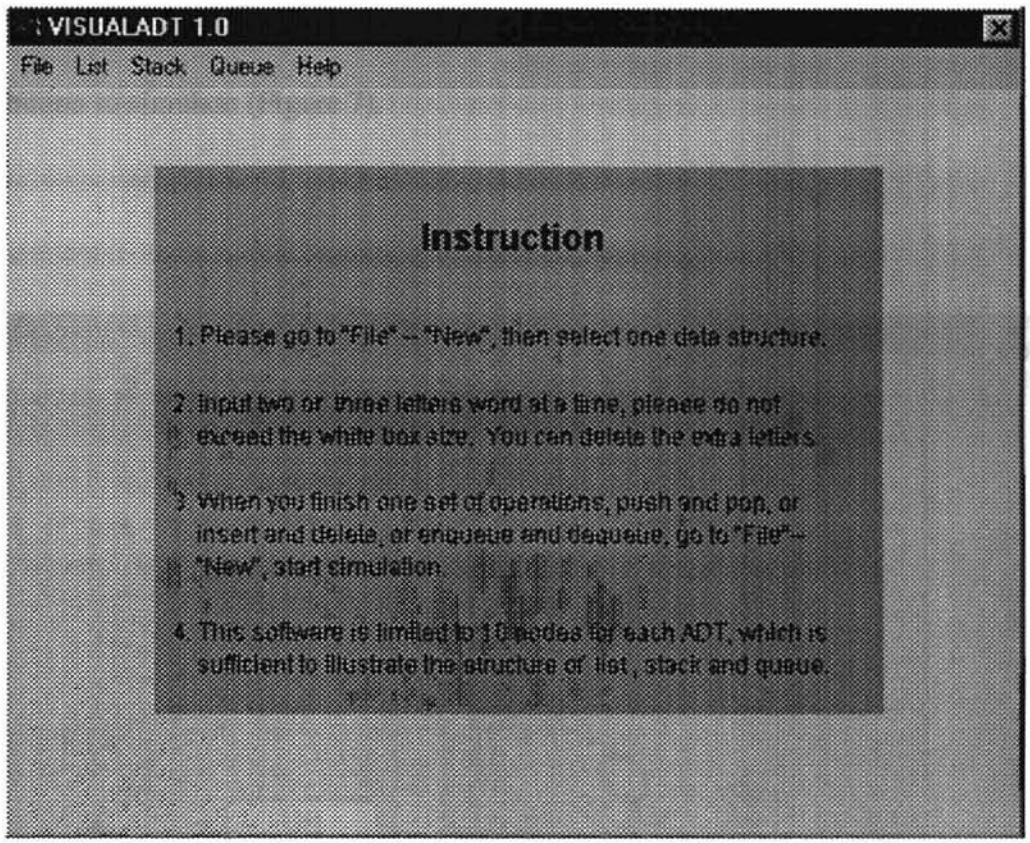


Figure 5: Instruction Window

b. At the beginning, the user is able to create one data structure from "File" menu, "New" option (Figure 6).

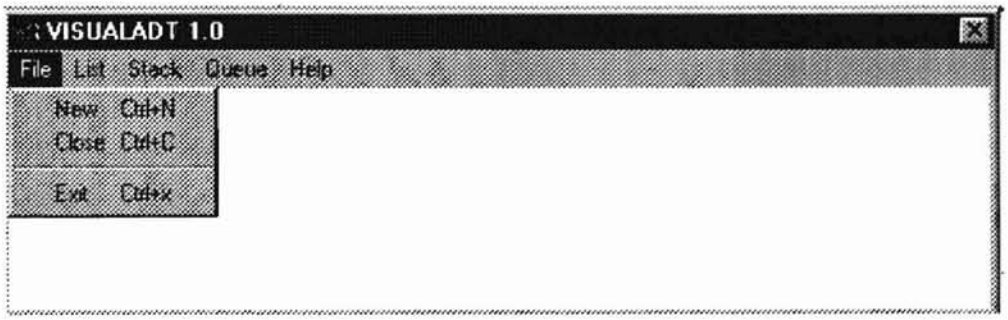


Figure 6: File Menu

c. The user is able to select one of the three data structures by clicking one button to simulate (Figure 7).

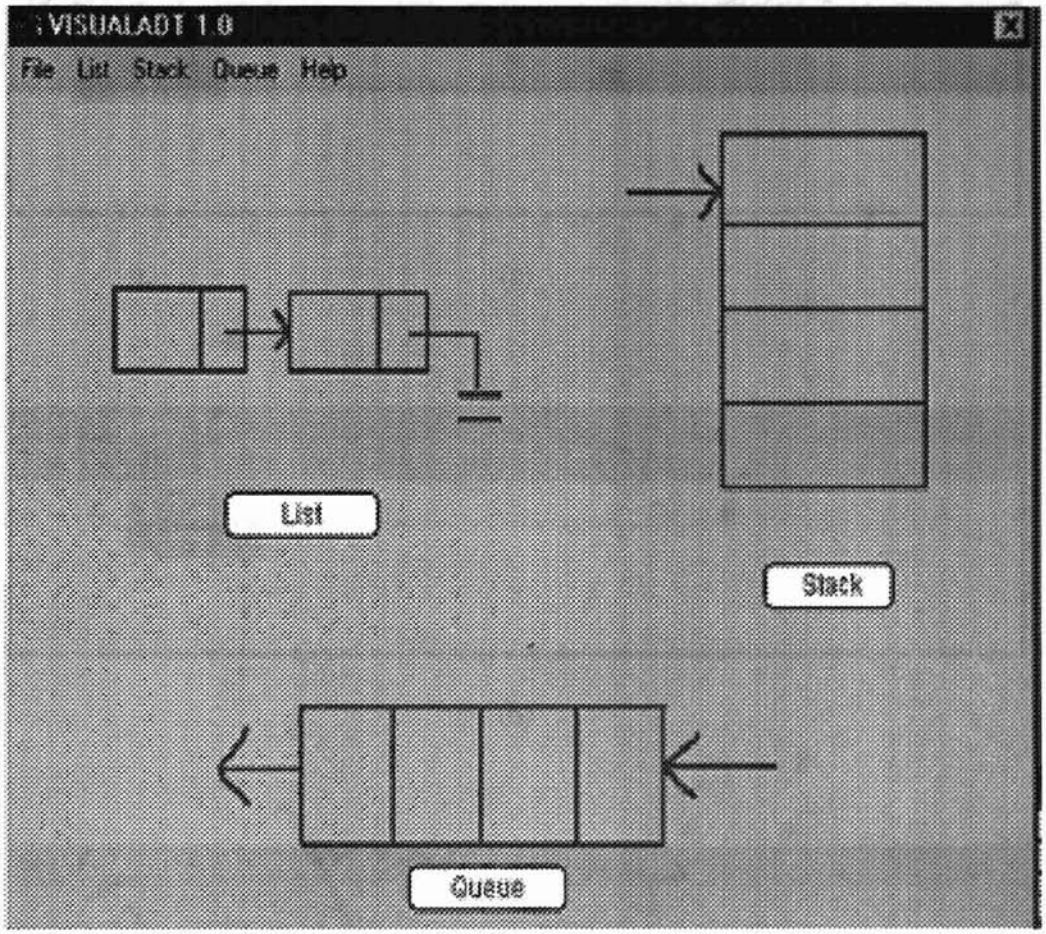
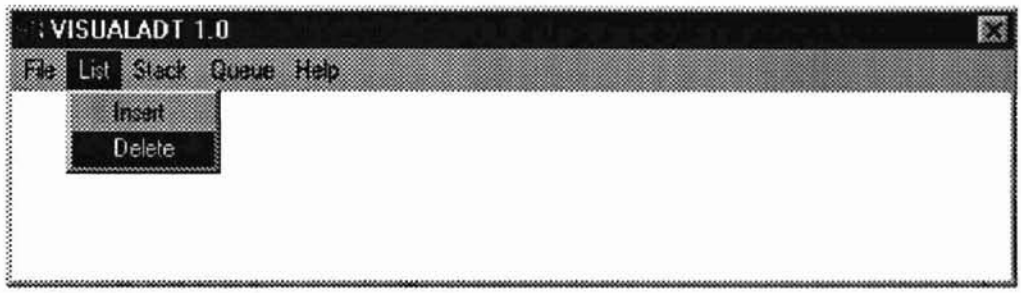


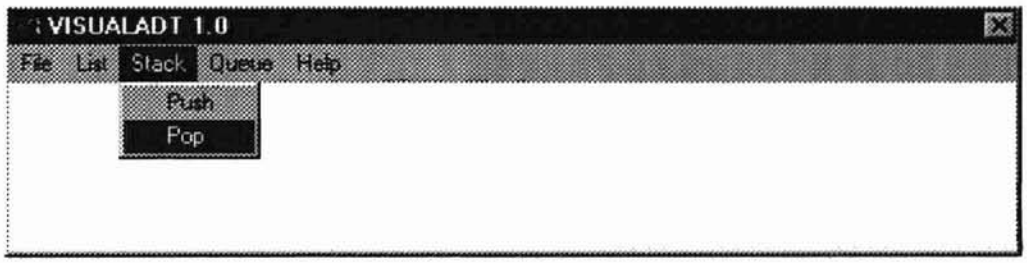
Figure 7: New Window

d. The user is able to execute operations by highlighting and clicking the operation from menu.

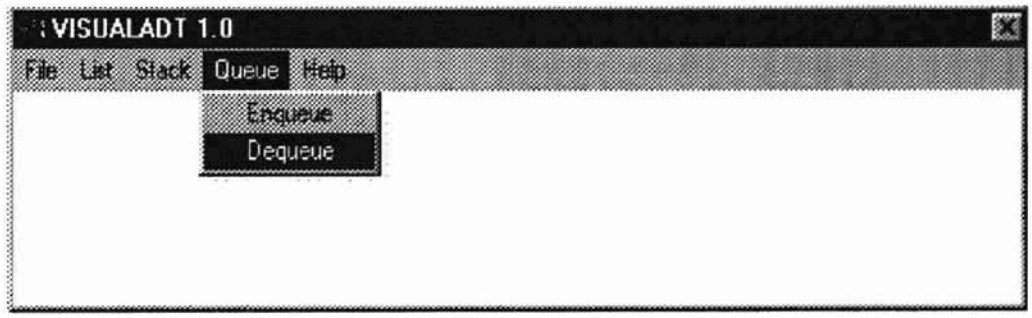
e. The user is able to perform the related operations on the selected data structure (Figure 8).



(a)



(b)



(c)

Figure 8: List, Stack and Queue Menu

f. The user is able to input the test data during the running time and see the result. Since the user selected the list ADT, the operations related to stacks and queues are disabled. This insures that the user performs correct operations (Figure 9).

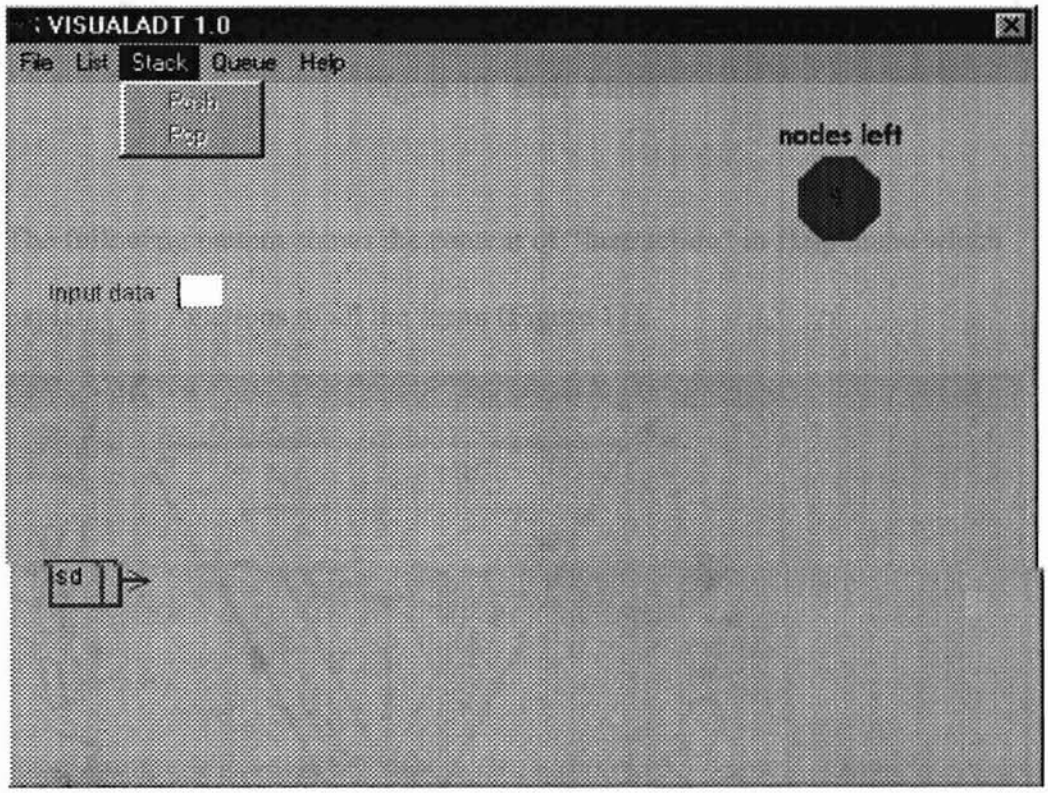


Figure 9: Run-time List with Disabled Stack and Queue Menu

g. For each of the three ADTs, the user is able to have at most 10 nodes.

h. A window in Help menu shows the instructions about how to use the VisualADT simulator (Figure 10).

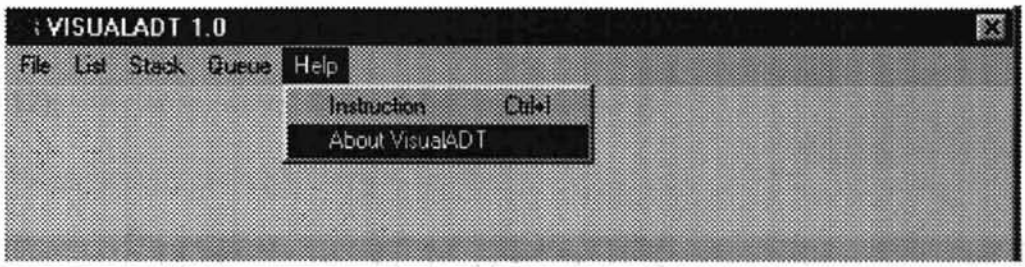


Figure 10: Help Menu

The following pattern shows the content of "Instruction" in Help menu which explains the functions of all the menu (Figure 11).

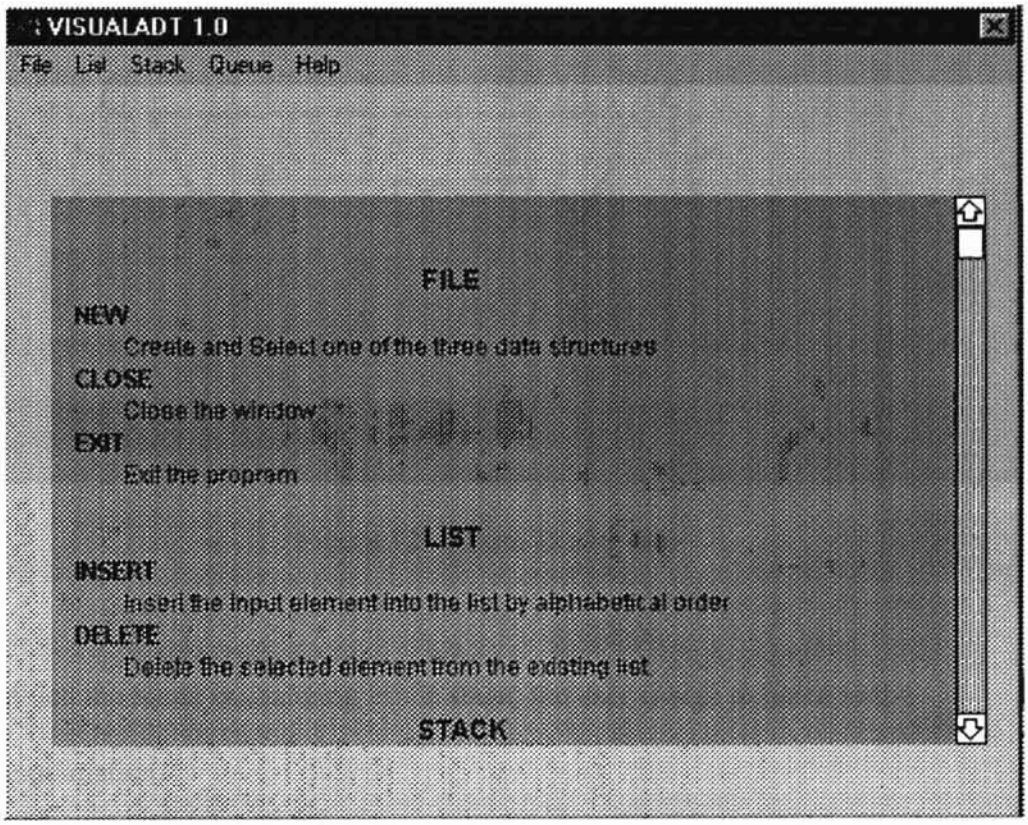


Figure 11: Help - Instruction

i. When the user performs some illegal operations, such as try to delete node from the empty list, an alarm dialog box pops-up with a system beep (Figure 12).

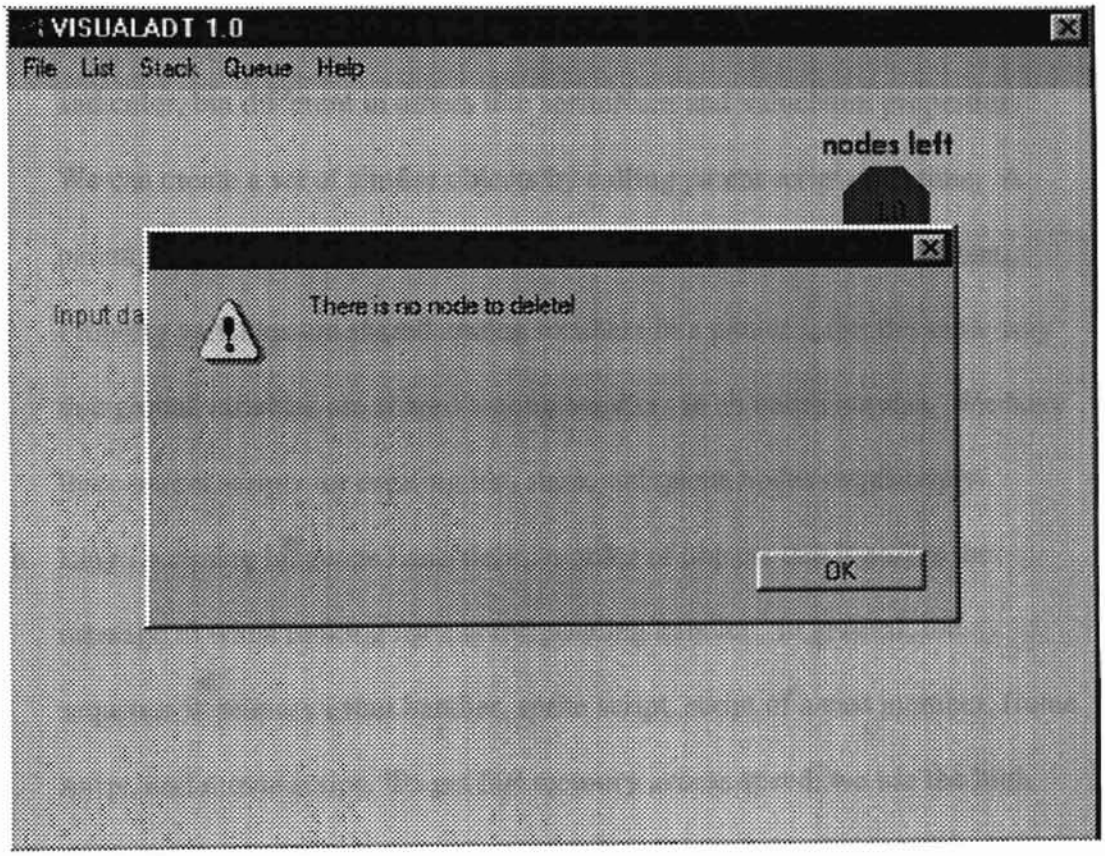


Figure 12: Alarm Dialog Box

j. When the scene is changing in the stage, the user is able to listen to the explanation.

4.5 Implementation Details

Correctness and efficiency are the most important concerns of the implementation of an education aided simulator. To ensure the correctness, memory efficiency, we made the following decisions:

a. Parent scripts and child scripts are the two important concepts. We create ten nodes for each data structure that are similar in some ways like same shape and color, but different in others like spriteNum and valueNum properties. We can create a set of similar objects by calling parent script each time. A parent script contains a set of handlers and property variable declarations. Property variables are shared among handlers in a parent script the same way that global variables are shared among handlers in an entire movies. We have three parent scripts for creating list, stack and queue nodes respectively.

b. Lingo scripting language has a definite order of objects that it passes the message to when looking for a corresponding handler. In general, the sequence is primary event handler, sprite script, script of a cast member, frame script, and movie script. To get fast memory access speed, we use the high priority script as much as we can.

c. Instead of using statements for each instance, we write a handler once and then call it from different places in the movie. In addition, when we revise the handler, we revise every instance where the handler is called. Also, we can place the handler's scripts in an external cast and use the cast in other movies.

d. Instead of creating the handlers and global variables for each object in a set of

similar objects, we use parent script and child script to avoid unnecessary allocation of the memory. The effectiveness of parent scripts comes from Lingo's ability to create multiple copies, or instances, of the script's content. Each child object is an instance. This also increases the memory efficiency.

e. Instead of using arrays, we use lists to keep track of and update a set of data. This provides efficient use of the storage space and saves memory access time. Director offers two types of lists:

   - Linear list, in which each element is a single value.
   - Property list, in which each element consists of a property and a value separated by a colon.

   Both linear and property lists can be unsorted or sorted in alphabetical order. Lingo can create, sort, add to, or reorder a list's elements.

f. Instead of using a self-defined list, we use a build-in list: the actorList. The actorList is a list of all child objects currently in the movie. We can clear child objects from the actorList by setting the actorList to empty. Then we can generate new child objects with the property we need.

g. We use the text of member property to display messages and record what the user types as input generator. We also can edit, test and set the field cast member property.

h. To get a smooth transition, we use a for loop to change the horizontal or vertical position of an object by one pixel a time. The user is able to watch the object moving to its destination.

i. We use an alert dialog box to provide error messages in the movie. The dialog box causes a system beep and displays a message which can contain up to 255 characters.

j. We create custom pull-down menus which give the user flexibility to choose the items provided. We define the menu in *on StartMovie* handler in a movie script, therefor it's available during the entire movie.

k. We use appropriate purge priority of the cast member if the memory is low. Purge priority has four levels.

Purge Priority	Description
3-Normal	The selected cast member will be removed from memory as necessary. This is the default.
2-Next	The selected cast member will be among the next to be removed from memory.
1-Last	The selected cast member will be among the last to be removed from memory.
0-Never	The selected cast member remains in memory and is never purged.

l. We use Astound Sound software version 2.01 to add sound through microphone. Astound Sound is developed for Gold Disk Inc. by NIME Enterprises Company. There is "Record Sound" option inside "File" menu of Astound Sound.

4.6 Director 5 Controls

I have used the following Director 5 Controls [29] [30] [31]:

a. Paint Window: It provides the same tools in a paint application such as Microsoft paint. It supports Photoshop filters and new tweenable filters for graphic effects. This creates and edits the user interface.

b. Cast Window: It is a multimedia database of graphics, text, sound effects, music and Lingo scripts. It contains all the information in a movie.

c. Score Window: It keeps track of each cast member on the stage in each frame of a movie and controls tempos and the timing of sounds, transitions, and palette changes.

d. Control Panel: It provides a set of controls similar to those on VCR. The user can use them to play, stop, or rewind a movie.

e. Stage Window: It is where the movie appears. It is always open.

f. Sound Control: The user can import the sounds and music into a movie and can control it with Lingo script language or a temp setting.

g. Lingo Script: It is director's scripting language that adds interactively to the multimedia project. It can combine animation and sound in ways that score alone can't.

4.7 Running the Simulator with Sample Inputs

a. The list data structure with 10 nodes with voice message "The list is full, please delete first." (Figure 13).

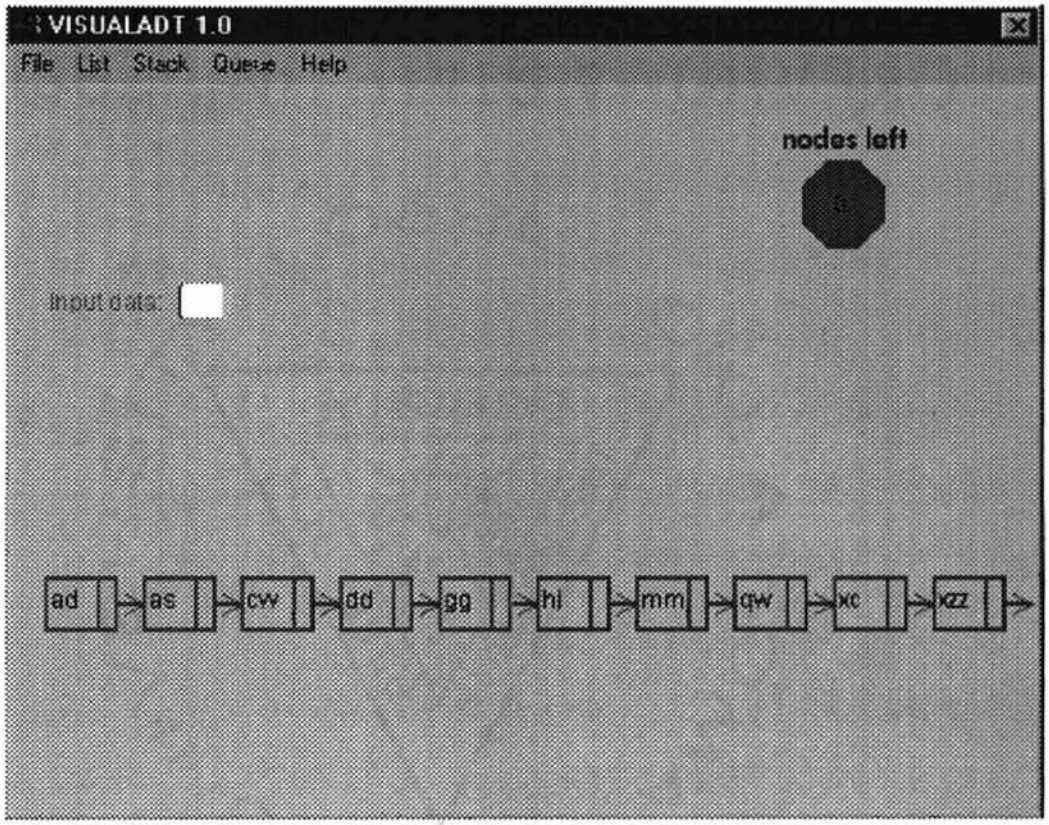


Figure 13: List ADT with 10 Nodes

b. The user input a data, "gg", select "Insert" operation from the "List" menu (Figure 14).

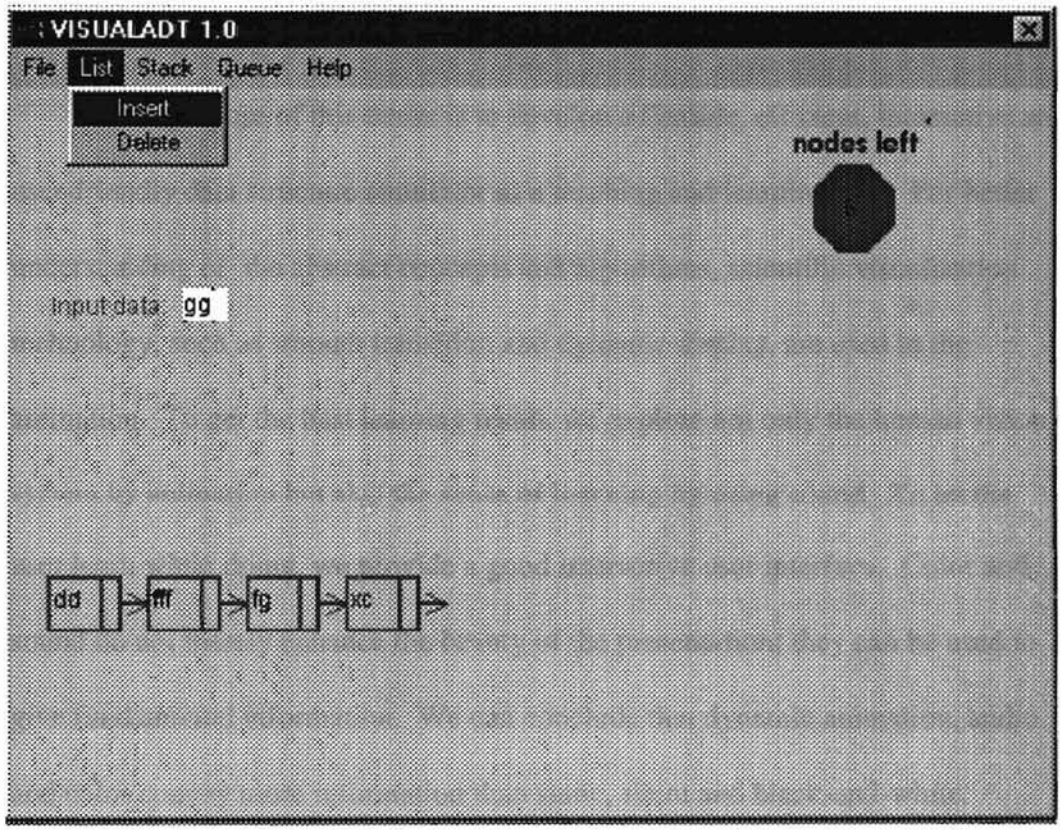


Figure 14: List ADT with Sample Input

CHAPTER V

SUMMARY AND FUTURE WORK

5.1 Summary

The purpose of this thesis is to develop a flexible, efficient, interactive and user-friendly data structure simulator as a teaching and learning tool. For better understanding of the abstract concepts and algorithms, scientific visualization technology, such as smooth transition and dynamic display, are used in the animation. To get the best learning result, we explore not only the human vision system by animation but also the sense of listening by using sound. To let the user learn while doing, we provide a good interactive user interface. Color and sound do not merely enhance the beauty of the presentation; they can be used to give fundamental information. We can conclude that dynamic animation, audio and color convey more information than static, silent and black-and-white.

This simulator is developed on Microsoft Windows 95 Operating System with Micromedia Director 5 software and Lingo scripting language as a 32-bit application which has good software reusability, maintainability and accessibility.

5.2 Future Work

In addition to these three linear data structures we animated, there are also many non-linear, tree-based data structures which are important in computer science. There will be a lot work to do in simulating the tree-based data structures and their related operations.

The VisualADT simulator has limited nodes for each data structure. The number of nodes can be unlimited by letting the user select which part of the simulator to be shown on the screen.

Also, we can put the simulator on the Internet with Shockwave plug-in as an Internet multimedia movie for distance learning.

BIBLIOGRAPHY

[1] Arra, S. K., "Object-Oriented Data Structure Animation", *M.S. Thesis*, Computer Science Department, Oklahoma State University, Stillwater, OK, 1992.

[2] Baecker, Ronald M., *Sorting Out Sorting*, 16mm color sound film, 25 minutes, Computer Science Department, University of Toronto, Toronto, ON, 1981.

[3] Baecker, R. M. and Buxton, W. A., *Reading in Human -Computer Interaction: A Multidisciplinary Approach*, Los Altos, CA: Morgan Kaufmann, 1987.

[4] Baskerville, David B., "Graphic Presentation of Data Structures in the DBX Debugger", *Report No. UCB/CSD 86/260*, Computer Science Department, University of California at Berkeley, CA, 1985.

[5] Bentley, Jon L. and Kernighan, Brian W., "A System for Algorithm Animation: Tutorial and User Manual", *Computer Science Technical Report,* No. 132, Murray Hill, NJ: AT&T Bell Laboratories, 1987.

[6] Blattner, M. M. and Dannenberg, R. B., *Multimedia Interface Design*, New York: ACM Press, 1992.

[7] Blattner, M. M. and Greenberg, D. A., "Earcons and Icons: Their Structure and Common Design Principles", *Human -Computer Interaction* 4(1): pp. 11-44, 1989.

[8] Bly, B. A., "Presenting Information in Sound", *In Proceedings of CHI '85 Conference on Human factors in Computing Systems*, pp. 371-375, New York: ACM Press, 1985.

[9] Booth, K. S., *PQ-Trees,* 16mm color silent film, 12 minutes, 1975.

[10] Brown, M. H., *Algorithm Animation*, Cambridge, MA: The MIT Press, 1988.

[11] Brown, G. P., Carling, R. T., Herot, C. F., Kramlich, D. A., and Souza, P., "Program Visualization: Graphical Support for Software Development", *IEEE Computer,* 18 (2): pp. 27-35, 1985

[12] Brown, M. H., Hershberger, J., *Color and Sound in Algorithm Animation*, Palo Alto, CA: Systems Research Center, Digital Equipment Corporation, 1991.

[13] Buxton, W., Bly, S. A., Frysinger, S. P. and Lunney, D., "Communications with Sound", *Proceedings of the ACM SIGCHI Human Factors in Computing Systems Conference,* pp. 115-119, New York: ACM Press, 1984.

[14] Dahl, O. J., Dijkstra, E. W., Hoare, C. A. R., *Structured Programming*, New York: Academic Press, 1972.

[15] Dahl, O. J., Nygaard, J., "SIMULA – an ALGOL-Based Simulation

Language", *ACM Communications* 9: pp. 671-678, 1966.

[16] DeBoer, James M., "A System for the Animation of Micro-PL/I Programs", *M.S. Thesis*, Department of Computer Science, University of Toronto, Toronto, ON, 1974.

[17] Duisberg, Robert A., "Constraint-Based Animation: Temporal Constraints in the Animus System", *Ph.D. Thesis*, Computer Science Department, University of Washington, Seattle, WA, 1986.

[18] Earley, J., "Toward an Understanding of Data Structures", *ACM Communications,* 14: pp. 617-627, 1971.

[19] Gaver, W. W., "The SonicFinder: An Interface that Uses Auditory Icons", *Human-Computer Interaction*, 4(1), 1989.

[20] Geschke, C. M., Morris, J. H., Satterwaite, E. H., "Early Experience with Mesa", *ACM Communications,* 20: pp. 540-553, 1977.

[21] House, W. C., *Interactive Computer Graphics Systems*, Princeton, NJ: Petrocelli Books Inc., 1982.

[22] Jones, C. V., *Visualization and Optimization*, Norwell, MA: Kluwer Academic Publishers, 1996.

[23] Kleyn, M. F., Gingrich, Paul C., "GraphTrace-Understanding Object-Oriented Systems Using Concurrently Animated Views", *OOPSLA' 88 Conference*, pp. 191-205, ACM/SIGPLAN, New York, 1988.

[24] Knowlton, K. C., "L6: Bell Telephone Laboratories Low-Level Linked List Language, Two Black and White Sound Films", Murray Hill, NJ: Bell

Telephone Laboratories, 1966.

[25] Knuth, D. E., *The Stanford GraphBase: A Platform for Combinatorial Computing,* Reading, MA: Addison-Wesley Publishing Co., 1993.

[26] Lee, Wilson, "An Implementation of A Data Structures Display System", *M.S. Thesis*, Computer Science Department, Oklahoma State University, Stillwater, OK, 1988.

[27] Liskov, B. H., Snyder, A., Atkinson, R. and Schaffert, C., "Abstraction Mechanisms in CLU", *ACM Communications,* 20: pp. 564-576, 1977.

[28] London, R. L., Duisberg, R. A., "Animating Programs Using Smalltalk", *Computer*, 18: pp. 61-71, 1985.

[29] *Macromedia Director Lingo Dictionary (Version 5)*, Macromedia, Inc., 1996.

[30] *Macromedia Director Using Director (Version 5)*, Macromedia, Inc., 1996.

[31] *Macromedia Director Using Lingo (Version 5)*, Macromedia, Inc., 1996.

[32] Macromedia Inc. WWW page,
URL: http://www.Macromedia.com/macromedia/pr/1996/dir5.html

[33] McCormick, B. H., DeFanti, T. A. and Brown, M. D., "Visualization in Scientific Computing", *Computer Graphics* 21(6): pp. 1-14, 1987.

[34] Moher, Thomas G., "PROVIDE: A Process Visualization and Debugging Environment", *Technical Report*, Computer Science Department, University of Illinois at Chicago, Chicago, IL, 1985.

[35] Myers, B. A., "INCENSE: A System for Displaying Data Structures",

Computer Graphics 17(3): pp. 115-125, 1983.

[36] Wirth, N., *Algorithms and Data Structures,* Englewood Cliffs, New Jersey: Prentice-Hall International, 1986.

[37] Reiss, Steven P., "Graphical Program Development with PECAN Program Development Systems", *ACM Transactions on Software Engineering,* 14(6): pp.849-1988.

[38] Shaw, M., Wulf, W. A., and London, R. L., "Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators", *ACM Communications,* 20: pp. 553-564, 1977.

[39] Shen, Hung-che, "A Visual Aid for the Learning of Tree-Based Data Structure", *M.S. Thesis*, Computer Science Department, Oklahoma State University, Stillwater, OK, 1994.

[40] Shimomura, T.and Isoda, S., "Linked-List Visualization for Debugging", *IEEE Software*, 17: pp. 44-51, 1991.

[41] Smith, H. F., *Data Structures: Form and Function*, San Diego, CA: Harcourt Brace Jovanovich, Inc., 1987.

[42] Sommerville, I., *Software Engineering(third edition)*, Reading, MA: Addison-Wesley Publishing Co., 1989.

[43] Stasko, John T., "TANGO: A Framework and System for Algorithm Animation", *IEEE Computer*, 23(2): pp. 27-39, 1990.

[44] Thomas, F. and Johnston, O., *Disney Animation: The Illusion of Life,* New York: Abbeville Press, 1984.

[45] University of Southern California WWW page,

URL: http://langevin.usc.edu/BST/

[46] Weiss, M. A., *Data structures and Algorithm Analysis in C*, Menlo Park, CA: Addison-Wesley Publishing Co., 1996.

[47] Wirth, N., *Systematic Programming: An Introduction,* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1973.

[48] Wirth, N., *Programming in Modula-2 (fourth edition)*, New York: Springer-Verlag Inc., 1988.

[49] Wulf, W. A., Shaw, M., Hilfinger, P. N. and Flon, L., *Fundamental Structures of Computer Science, Reading*, MA: Addison-Wesley Publishing Inc., 1981.

[50] Yarwood, Edward, "Towards Program Illustration", *M.S. Thesis*, Department of Computer Science, University of Toronto, Toronto, ON, 1974.

[51] Zernik, D., Snir M. and Malki, D., "Using Visualization Tools to Understand Concurrency", *IEEE Software*, 18: pp. 87-92, 1992.

APPENDIX SOURCE CODE

Main Script

```
on startMovie
  -- Main script
  installMenu 2

  puppetSound "welcome5"
  updateStage

  -- global variable declarations
  global nodeList,maxNodes,firstNode,counter,queList,stackList
  global popCount,queCount,v1,linkList,insertPosition,deletePos
  global stackFlag,queFlag,delCount,index,delList,stackCount
  global listFlag,nodePos,linkCount,tempList

  set maxNodes to 10
  set firstNode to 29
  set counter to -1
  set stackList to [:]
  set popCount to 0
  set v1 to 0
  set deletePos to 0
  set insertPosition to 0
  set queList to [:]
  set delList to [:]
  set linkList to [:]
  set tempList to [:]
  set nodeList to []
  set queCount to 1
  set stackCount to 29
  set stackFlag to TRUE
  set queFlag to TRUE
  set listFlag to TRUE
  set delCount to 0
  set nodePos to 0
  set index to TRUE
```

```
  set linkCount to FALSE

  put "" into field "Word"
  set the stageColor to 43

  repeat with i = firstNode to (maxNodes + firstNode - 1)
    -- nodes 0 - 9 are sprites 29 - 38
    puppetSprite i, TRUE
  end repeat
  repeat with i = (maxNodes + firstNode) to (maxNodes * 2 + firstNode -1)
    -- values are sprites 39 to 48
    puppetSprite i, TRUE
    set the text of member (38 - maxNodes - firstNode + i) to ""
  end repeat

  repeat with i =29 to 48
    set the backColor of sprite i to 43
  end repeat

  --alarm box if the user didn't select a data structure
  set the script of menuItem "Insert" of menu "List" to "listHandler"
  set the script of menuItem "Enqueue" of menu "Queue" to "listHandler"
end startMovie

--stack push handler
on stackPush Token
  global nodeList,root,maxNodes,v1

  set newToken to pushToken(root,Token)
end stackPush

--enque handler
on Enq Token
  global nodeList,root,maxNodes,v1,delList

  set newToken to pushToken(root,Token)

  --check if newToken is blank or a space
  if newToken<>"" or newToken<>" " then
    addProp(delList,newToken,v1)
  end if
end Enq

--list insert handler
on Insert Token
```

```
  global root

  set newToken to pushToken(root,Token)
end Insert
```

Menu Definition

```
menu: File
New/Nl go to frame "New"
Close/Cl go to frame "Close"
   (-
Exit/Xl go to frame "Exit"
menu: List
Insertl go to frame "Insert"
Deletel go to frame "Delete"
menu: Stack
Pushl go to frame "Push"
Popl go to frame "Pop"
menu: Queue
Enqueuel go to frame "Enque"
Dequeuel go to frame "Deque"
menu: Help
Instruction/Il go to frame "Ins"
About VisualADTl go to frame "About"
```

Start Script

```
on enterFrame
  --move the welcome window and instruction window into or
  --off the stage
  puppetSprite 6, TRUE
  puppetSprite 5, TRUE

  set the locV of sprite 5 to the stageBottom
  set the locH of sprite 5 to 74

  startTimer
  repeat while the timer <6*60
    nothing
  end repeat

  --move the welcome window
  repeat with i = 1 to 350
```

```
  set the locV of sprite 6 to (the locV of sprite 6 - 1)
  updateStage
 end repeat

 puppetSound "instruc"
 updateStage

 --move the instruction window
 repeat with i = 1 to 430
  set the locV of sprite 5 to (the locV of sprite 5 - 1)
  updateStage
 end repeat

 startTimer
 repeat while the timer <8*60
  nothing
 end repeat
 repeat with i = 1 to 350
  set the locV of sprite 5 to (the locV of sprite 5 - 1)
  updateStage
 end repeat

 puppetSound "new"
 updateStage
 startTimer
 repeat while the timer <5*60
  nothing
 end repeat
end

on exitFrame
 puppetSprite 6, FALSE
 puppetSprite 5, FALSE

 pause
end
```

New Handler

```
on enterFrame
 global firstNode,maxNodes,h,nodeList
 repeat with i = firstNode to (maxNodes + firstNode - 1)
  -- nodes 0 - 9 are sprites 29 - 38
  puppetSprite i, TRUE
 end repeat
```

```
  repeat with i = (maxNodes + firstNode) to (maxNodes * 2 + firstNode -1)
    -- values are sprites 39 to 48
    puppetSprite i, TRUE
    set the text of member (38 - maxNodes - firstNode + i) to ""
  end repeat

  repeat with j=(firstNode+maxNodes) to 48
    set h to the locH of sprite j
    set v to the locV of sprite j
    --move the valuebox out the stage
    repeat with i=firstNode to (firstNode+maxNodes-1)
      set the visible of sprite i to FALSE
      set the locH of sprite (i+maxNodes) to (h+600)
      set the locV of sprite (i+maxNodes) to (v+600)
    end repeat
  end repeat

  updateStage
end

on exitFrame
  global firstNode,maxNodes,stackList,nodeList,counter,stackCount
  global queCount,popCount,tempList,queList,delList,delCount
  global insertPosition,index,linkCount,stackFlag,queFlag,listFlag
  global v1,deletePos,linkList,nodePos

  set counter to -1
  put 10 into field "Que Nodes left"
  put 10 into field "Nodes left"
  set stackCount to 29
  set stackList to [:]
  set nodeList to []
  set tempList to [:]
  set queList to [:]
  set delList to [:]
  set delCount to 0
  set popCount to 0
  set queCount to 1
  set insertPosition to 0
  set the actorList to []
  set index to TRUE
  set linkCount to FALSE
  set stackFlag to TRUE
  set queFlag to TRUE
  set listFlag to TRUE
```

```
  set v1 to 0
  set deletePos to 0
  set linkList to [:]
  set nodePos to 0

  pause
end
```

"File-Close" Menu Script

```
on enterFrame
  -- "File-Close" script
  global firstNode,maxNodes,h,nodeList
  repeat with i = firstNode to (maxNodes + firstNode - 1)
    -- nodes 0 - 9 are sprites 29 - 38
    puppetSprite i, TRUE
  end repeat
  repeat with i = (maxNodes + firstNode) to (maxNodes * 2 + firstNode -1)
    -- values are sprites 39 to 48
    puppetSprite i, TRUE
    set the text of member (38 - maxNodes - firstNode + i) to ""
  end repeat

  --set the nodes out the stage
  repeat with j=(firstNode+maxNodes) to 48
    set h to the locH of sprite j
    set v to the locV of sprite j

    repeat with i=firstNode to (firstNode+maxNodes-1)
      set the visible of sprite i to FALSE
      set the locH of sprite (i+maxNodes) to (h+600)
      set the locV of sprite (i+maxNodes) to (v+600)

    end repeat
  end repeat
end

on exitFrame
  pause
end
```

Push Frame Script

```
on enterFrame
  global root,nodeList,firstNode,maxNodes,stackCount,stackFlag
```

```
 set stackCount to firstNode

 if count(stackList)=0 then
  set the locH of sprite firstNode to 250
  set the locV of sprite firstNode to (the stageTop-140)
  set the locH of sprite (maxNodes + firstNode) to 245
  set the locV of sprite (maxNodes + firstNode) to (the stageTop-150)
  set stackFlag to TRUE
  set root to new(script"Stack Parent Script",0,0,0,0,0)
  add nodeList,root
 end if

 set the script of menuItem "Push" of menu "Stack" ¬
 to "pushHandler"

 updateStage
end

on exitFrame
 pause
end
```

Push Handler

```
on pushHandler
   set Token to field "Word"
   put "" into field "Word"
   stackPush(Token)

end pushHandler
```

Stack Parent Script

```
--Parent script for creating a stack node and value box
property  level, number, Boxes, Values, valueList, spriteNum, valueNum

on new me, nodeNumber, oldLevel, cloneNum, cloneNode, leftChild
 global nodeList, maxNodes, firstNode,counter,stackFlag
 if count (nodeList) >= maxNodes then
  puppetsound "staful"
  updateStage

  repeat with i = 1 to 4
   beep (1)
   put "no" into field "Nodes left"
```

```
    updateStage
    startTimer
    repeat while the timer < 30
      nothing
    end repeat
    put "" into field "Nodes left"
    updateStage
    startTimer
    repeat while the timer < 30
      nothing
    end repeat
  end repeat
  put "0" into field "Nodes left"
  abort
 end if
 set number to nodeNumber
 set level to oldLevel
 set spriteNum to (nodeNumber + firstNode)
 set valueNum to (nodeNumber + 38)
 if stackFlag=TRUE then
  set Boxes to new (script "stackNode Parent" , count(nodeList), cloneNum)
  set Values to new (script "stackValue Parent", count(nodeList), cloneNum)
 else
  set Boxes to new (script "stackNode Parent" ,(count(stackList)-1), cloneNum)
  set Values to new (script "stackValue Parent",(count(stackList)-1), cloneNum)
 end if

 set valueList to [:]
 set the visible of sprite (firstNode+counter) to TRUE
 return me
end

on pushToken me, Token
 global nodeList, root, newNode,v1,stackList,firstNode,maxNodes

 set newToken to Token

 if newToken = "" or newToken=" " then
  alert"Please input data!"
  puppetSound "input"
  updateStage
  startTimer
  if level = 0 then set i to 4
  repeat while the timer < i * 60
   nothing
```

```
    end repeat
  else
    puppetSound "push"
    updateStage

    --check if the stack has more than ten nodes
    if count(stackList)<10 then
      addProp (stackList,newToken,v1)
    end if

    if count(stackList)<=1 then
      set the text of member valueNum to getPropAt(stackList,1)

      repeat with i = 1 to (the stageTop+10)
        set the locV of sprite 29 to ¬
          (the locV of sprite 29 + 1)
        set the locV of sprite 39 to ¬
          (the locV of sprite 39 + 1)
        updateStage
      end repeat
      put maxNodes - count (nodeList) into field "Nodes left"
      -- Show the insert
      updateStage
    else
      set newToken to splitNode (me,newToken,v1)
    end if

  end if

  updateStage
  return newToken
end pushToken


on splitNode me, Token, loc
  global nodeList, maxNodes, firstNode, newNode,stackList,v1


  set newNode to new (script "Stack Parent Script", count(nodeList),level,number, me,
v1)

  add (nodeList, newNode)
  set the text of member 38 to getPropAt(stackList,1)
  --add the values to the display sprite for me
```

```
  set the text of member the valueNum of newNode to
getPropAt(stackList,count(stackList))

  -- Move new node into position
  if level = 0 then
    moveDown getAt(nodeList,1)
  end if
  put maxNodes - count (nodeList) into field "Nodes left"
  startTimer
  if level = 0 then set i to 1
  repeat while the timer < i * 60
    nothing
  end repeat

  return newToken
end splitNode

on moveDown me
  global maxNodes

  repeat with i = 29 to (29+(count(stackList)-2))
    repeat with j = 1 to 30
      set the locV of sprite i to ¬
        the locV of sprite i + 1
      set the locV of sprite (i+maxNodes) to ¬
        the locV of sprite (i+maxNodes) + 1
      updateStage
    end repeat
  end repeat
end moveDown
```

Pop Frame Script

```
on enterFrame
  global nodeList,firstNode,maxNodes,newNode,counter,stackCount,spriteNum
  global stackFlag

  set stackFlag to FALSE
  if count(stackList)=0 then
    alert "There is no node to pop!"
    beep
  else
    puppetSound "pop"
    updateStage
    --move the top node out
```

```
  repeat with i = 1 to (the stageTop+300)
    set the locV of sprite (stackCount+maxNodes+count(stackList)-1) to  (the locV of
sprite (stackCount+maxNodes+count(stackList)-1) - 1)
    set the locV of sprite (stackCount+count(stackList)-1) to ¬
       (the locV of sprite (stackCount+count(stackList)-1) - 1)
    updateStage
  end repeat

  deleteAt(stackList,count(stackList))
  deleteAt(nodeList,count(nodeList))

  --move the rest node up one location
  repeat with j=0 to count(stackList)
    repeat with i=1 to 30
      set the locV of sprite (stackCount+j) to (the locV of sprite (stackCount+j)-1)
      set the locV of sprite (stackCount+maxNodes+j) to (the locV of sprite
(stackCount+j)-5)
      updateStage
    end repeat
  end repeat

  put (maxNodes-count(nodeList)) into field "Nodes left"
  put "" into field "Word"
  updateStage
  startTimer
  repeat while the timer <2*30
    nothing
  end repeat

  if count(stackList)=0 then
    puppetSound "staemp"
    updateStage
    startTimer
    repeat while the timer < 7*30
      nothing
    end repeat
  end if

 end if
end

on exitFrame
 go to "Push"
end
```

Stack Node Box Parent

```
-- Create new stack node box
property  valueList

on new me, nodeNumber, cloneNode
 global firstNode,counter,stackFlag
 set spriteNum to (nodeNumber + firstNode)
 set cloneSprite to (cloneNode + firstNode)
 set the castNum of sprite spriteNum to the number of cast "stackNode"
 if stackFlag=TRUE then
  if count(nodeList)=0 then
   set the locH of sprite spriteNum to the locH of sprite cloneSprite
   set the locV of sprite spriteNum to the locV of sprite cloneSprite
  else
   set the locH of sprite spriteNum to (the locH of (sprite (spriteNum-1)))
   set the locV of sprite spriteNum to ((the locV of sprite (spriteNum-1)))
  end if
 else
  set the locH of sprite spriteNum to 250
  set the locV of sprite spriteNum to  86
 end if

 set counter to (counter+1)

 return me
end
```

Stack Value Parent

```
-- create a new stackValue slot
on new me, nodeNumber, cloneNode
 global maxNodes, firstNode,stackFlag
 set i to nodeNumber + firstNode + maxNodes
 set valueNum to nodeNumber + 38
 set cloneSprite to cloneNode + firstNode + maxNodes
 set the castNum of sprite i to valueNum
 if stackFlag=TRUE then
  if count(nodeList)=0 then
   set the locH of sprite i to the locH of sprite cloneSprite
   set the locV of sprite i to the locV of sprite cloneSprite
  else
   set the locH of sprite i to (the locH of sprite (i-1))
   set the locV of sprite i to (the locV of sprite (i-1))
  end if
```

```
  else
    set the locH of sprite i to  245
    set the locV of sprite i to  76
  end if

end
```

Enqueue Frame Script

```
on enterFrame
  global root,nodeList,firstNode,maxNodes

  if count(nodeList)=0 then
    set stageWidth to (the stageRight- the stageLeft)
    set the locH of sprite firstNode to (stageWidth+32)
    set the locV of sprite firstNode to 230
    set the locH of sprite (maxNodes + firstNode) to (stageWidth+21)
    set the locV of sprite (maxNodes + firstNode) to 220
    set root to new(script"Queue Parent Script",0,0,0,0,0)
    set the text of member 38 to " "
  end if

  set the script of menuItem "Enqueue" of menu "Queue" ¬
  to "enqHandler"

  updateStage
end

on exitFrame
  pause
end
```

Deque Frame Script

```
on enterFrame
  global nodeList,firstNode,maxNodes,newNode
  global counter,popCount,queCount,delCount,queFlag

  set queFlag to FALSE
  if count(queList)=0 then
    alert "There is no node to dequeue!"
    beep
  else
    puppetSound "deq"
    updateStage
```

```
    repeat with i = 1 to (the stageLeft+290+33*(count(nodeList)))
      set the locH of sprite (the spriteNum of getAt(nodeList,1)) to ¬
        (the locH of sprite (the spriteNum of getAt(nodeList,1))-1)
      set the locH of sprite (the valueNum of getAt(nodeList,1)+1) to ¬
        (the locH of sprite (the valueNum of getAt(nodeList,1)+1)-1)
      updateStage
    end repeat

    repeat with i=2 to count(nodeList)
      repeat with j=1 to 32
        set the locH of sprite (the spriteNum of getAt(nodeList,i)) to ( the locH of sprite (the
spriteNum of getAt(nodeList,i))-1)
        set the locH of sprite (the valueNum of getAt(nodeList,i)+1) to (the locH of sprite
(the valueNum of getAt(nodeList,i)+1)-1)
        updateStage
      end repeat
    end repeat

    deleteAt(queList,1)
    deleteAt(nodeList,1)
    put (maxNodes-count(nodeList)) into field "Que Nodes left"
    set counter to (counter-1)
    set popCount to (popCount+1)

    updateStage
    startTimer
    repeat while the timer < 4*30
      nothing
    end repeat

    if count(queList)=0 then
      puppetSound "queemp"
      updateStage
      startTimer
      repeat while the timer < 8*30
        nothing
      end repeat
    end if

  end if
end

on exitFrame
  go to "Enque"
```

```
end
```

Enque Handler

```
on enqHandler
  set Token to field "Word"
  put "" into field "Word"
  Enq(Token)

end enqHandler
```

Queue Parent Script and pushToken Handler

```
--Parent script for creating a queue node
property  level, number, Boxes, Values, valueList, spriteNum, valueNum

on new me, nodeNumber, oldLevel, cloneNum, cloneNode, leftChild
  global nodeList, maxNodes, firstNode,counter,delCount,queFlag
  if count (nodeList) >= maxNodes then
    puppetSound "queful"
    updateStage

    repeat with i = 1 to 4
      beep (1)
      put "no" into field "Que Nodes left"
      updateStage
      startTimer
      repeat while the timer < 30
        nothing
      end repeat
      put "" into field "Que Nodes left"
      updateStage
      startTimer
      repeat while the timer < 30
        nothing
      end repeat
    end repeat
    put "0" into field "Que Nodes left"
    abort
  end if
  set number to nodeNumber
  set level to oldLevel
  set spriteNum to (nodeNumber + firstNode)
  set valueNum to (nodeNumber + 38)
  if queFlag=TRUE then
```

```
  set Boxes to new (script "queNode Parent", count(nodeList), cloneNum)
  set Values to new (script"queValue Parent",count(nodeList), cloneNum)
 else
  set Boxes to new (script "queNode Parent", delCount, cloneNum)
  set Values to new (script "queValue Parent",delCount, cloneNum)
 end if

 set valueList to [:]
 set the visible of sprite spriteNum to TRUE
 return me
end

on pushToken me, Token
 global nodeList,root,Order,v1,newToken,maxNodes,queFlag,delCount,delList

 set newToken to Token
 if newToken = "" or newToken=" " then
  alert"Please input data!"
  puppetSound "input"
  updateStage
 else
  if count(queList)<10 then
   puppetSound "enq"
   updateStage

   addProp (queList, newToken, v1)
   --addProp(delList,newToken,v1)
   if queFlag=FALSE then
    set delCount to count(delList)
    if count(delList)>=10 then
     set delCount to (count(delList) mod 10)
    end if
   end if
  end if

  if count(queList)<=1 then
   -- put the values into the sprite
   set the text of member valueNum to getPropAt (queList,1)

   add nodeList,root
   put maxNodes - count (nodeList) into field "Que Nodes left"
   --move the firstNode and value box out of stage
   repeat with i = 1 to (the stageLeft+290)
    set the locH of sprite 29 to ¬
     (the locH of sprite 29 - 1)
```

```
      set the locH of sprite 39 to ¬
        (the locH of sprite 39 - 1)
      updateStage
    end repeat
    -- Show the insert
    updateStage
  else
    set newToken to splitNode (me, newToken,v1)
  end if

 end if

 updateStage
 return newToken
end pushToken

on splitNode me, Token, loc
 global nodeList, Order, maxNodes, firstNode,newNode,v1,delCount,queFlag

 if queFlag=TRUE then
   set newNode to new (script "Queue Parent Script",count(nodeList), level, number, me,
v1)
 else
   set newNode to new (script "Queue Parent Script",delCount,level, number, me, v1)
 end if

 updateStage
 add (nodeList, newNode)
 put maxNodes - count (nodeList) into field "Que Nodes left"

 -- add the values to the display sprite for newNode
 set the text of cast the valueNum of newNode to getPropAt (queList,count(queList))

 -- Move new node into position
 moveLeft newNode

 startTimer
 repeat while the timer < 1 * 60
   nothing
 end repeat

 return newToken
end splitNode

on moveLeft me
```

```
  global maxNodes
  set valueSprite to spriteNum + maxNodes
  repeat with i = 1 to (the stageLeft+290-33*(count(nodeList)-1))
   set the locH of sprite spriteNum to ¬
       (the locH of sprite spriteNum - 1)
   set the locH of sprite valueSprite to ¬
       (the locH of sprite valueSprite - 1)
   updateStage
  end repeat
end moveLeft
```

Queue Value Parent Script

```
--create a new queue value slot

on new me, nodeNumber, cloneNode
  global maxNodes, firstNode,delCount
  set stageWidth to (the stageRight- the stageLeft)
  if delCount>=10 then
   set i to (delCount mod 10)+firstNode + maxNodes
  else
   set i to nodeNumber + firstNode + maxNodes
  end if

  set valueNum to nodeNumber + 38
  set cloneSprite to cloneNode + firstNode + maxNodes
  set the castNum of sprite i to valueNum
  set the locH of sprite i to (stageWidth+21)
  set the locV of sprite i to 220
end
```

Queue Node Parent Script

```
-- Create new que node box
property  valueList

on new me, nodeNumber, cloneNode
  global firstNode,counter
  set stageWidth to (the stageRight- the stageLeft)
  set spriteNum to (nodeNumber + firstNode)
  set cloneSprite to (cloneNode + firstNode)
  set the castNum of sprite spriteNum to the number of cast "queNode"
  set the locH of sprite spriteNum to (stageWidth+32)
  set the locV of sprite spriteNum to 230
  set counter to (counter+1)
```

```
  return me
end
```

List Insert Frame Script

```
on enterFrame
  global root,nodeList,firstNode,maxNodes,listFlag,linkCount,linkList
  global tempList,v1

  if count(linkList)=0 then
    set the locH of sprite firstNode to -30
    set the locV of sprite firstNode to 260
    set the locH of sprite (maxNodes + firstNode) to -49
    set the locV of sprite (maxNodes + firstNode) to 250

    set root to new(script"List Parent Script",0,0,0,0,0)
    add the actorList,root
    set the text of member 38 to " "
    add nodeList,root
  end if

  set the script of menuItem "Insert" of menu "List" ¬
  to "insertHandler"

  updateStage
end

on exitFrame
  pause
end
```

List Parent Script and pushToken Handler

```
--Parent script for creating a list node
property  level, number, Boxes, Values, valueList, spriteNum, valueNum

on new me, nodeNumber, oldLevel, cloneNum, cloneNode, leftChild
  global nodeList, maxNodes, firstNode,counter,listFlag,nodePos,linkCount
  global deletePos

  if count (nodeList) >= maxNodes then
    puppetSound "listful"
    updateStage

    repeat with i = 1 to 4
```

```
    beep (1)
    put "no" into field "Que Nodes left"
    updateStage
    startTimer
    repeat while the timer < 30
     nothing
    end repeat
    put "" into field "Que Nodes left"
    updateStage
    startTimer
    repeat while the timer < 30
     nothing
    end repeat
   end repeat
   put "0" into field "Que Nodes left"
   abort
  end if
  set number to nodeNumber
  set level to oldLevel

  set spriteNum to (nodeNumber + firstNode)
  set valueNum to (nodeNumber + 38)

  set Boxes to new (script "listNode Parent", count(nodeList), cloneNum)
  set Values to new (script "listValue Parent", count(nodeList), cloneNum)

  if listFlag=TRUE then
   set the visible of sprite (firstNode+counter) to TRUE
  else
   set the visible of sprite (firstNode+nodePos-1) to TRUE
   set the visible of sprite (firstNode+counter) to TRUE
  end if

  return me
end

on pushToken me, Token
  global nodeList,root,Order,newNode,linkList,v1,tempList,linkCount
  global maxNodes,counter,firstNode

  if linkCount=TRUE then
   set tempList to [:]
   set the actorList=[]
   set counter to -1
   if count(linkList)<>0 then
```

```
    repeat with i=29 to 38
      set the locH of sprite i to -20
      set the locV of sprite i to -10
      set the text of member (i+maxNodes-1) to ""
      set the locH of sprite (i+maxNodes) to -20
      set the locV of sprite (i+maxNodes) to -10
      updateStage
    end repeat
    repeat with i=1 to count(linkList)
      addProp(tempList,getPropAt(linkList,i),v1)
      add the actorList, new (script "List Parent Script",(i-1), level, number, me, v1)
      set the locH of sprite (i+29-1) to 40+49*(i-1)
      set the locV of sprite (i+29-1) to 260
      set the locH of sprite (i+38) to 21+49*(i-1)
      set the locV of sprite (i+38) to 250
      set the text of member (38+i-1) to getPropAt(linkList,i)
      updateStage
    end repeat
  end if

  set linkCount to FALSE
end if

set newToken to Token

if newToken = "" or newToken=" "  then
  alert"Please input data!"
  puppetSound "input"
  updateStage
  startTimer
  if level = 0 then set i to 4
  repeat while the timer < i * 60
    nothing
  end repeat

else
  if count(linkList)<1 then
    puppetSound "insert2"
    updateStage
    addProp (linkList,newToken,v1)
    put linkList
    sort linkList
    addProp(tempList,newToken,v1)
    sort tempList
    -- put the values into the sprite
```

```
    set the text of member valueNum to getPropAt (linkList,1)

    repeat with i = 1 to 70
      set the locH of sprite firstNode to (the locH of sprite firstNode+1)
      set the locH of sprite (maxNodes + firstNode) to ((the locH of sprite (maxNodes +
firstNode))+ 1)
      updateStage
    end repeat
    put maxNodes-count(nodeList) into field "Que Nodes left"

    -- Show the insert
    updateStage
    return ""
   else
    repeat with i=1 to count(linkList)
     if getPropAt (linkList,i) = Token then
      alert"Please input a new data!"
      puppetSound "inputnew"
      updateStage
      startTimer
      if level = 0 then set i to 4
      repeat while the timer < i * 60
       nothing
      end repeat

      exit
     end if
    end repeat
    puppetSound "insert2"
    updateStage

    set newToken to splitNode (me, newToken, v1)
   end if

  end if

  updateStage
  return newToken
end pushToken

on splitNode me, Token, loc
  global nodeList, Order, maxNodes, firstNode, insertPosition
  global newNode,v1,linkList,tempList

  set newNode to new (script "List Parent Script", count(nodeList), level, number, me, v1)
```

```
  add the actorList,newNode
  add (nodeList, newNode)
  put maxNodes - count (nodeList) into field "Que Nodes left"

  addProp(linkList,Token,v1)
  put linkList
  sort linkList
  addProp(tempList,Token,v1)
  sort tempList

  repeat with i=1 to count(linkList)
   set the text of member (38+i-1) to getPropAt(linkList,i)
   if getPropAt (linkList,i) = Token then
    set insertPosition to i
   end if

  end repeat

  -- Move inserted node into position
  set insertNode to getAt(nodeList,insertPosition)
  if insertPosition >1 then
   set the locH of sprite (firstNode+insertPosition-1) to 40
   set the locV of sprite (firstNode+insertPosition-1) to 260
   set the locH of sprite (maxNodes + firstNode+insertPosition-1) to 21
   set the locV of sprite (maxNodes + firstNode+insertPosition-1) to 250
  end if

  moveRight insertNode

  startTimer
  if level = 0 then set i to 1
  repeat while the timer < i * 60
   nothing
  end repeat

  return newToken
end splitNode

on moveRight me
  global maxNodes,insertPosition,firstNode,nodeList
  set valueSprite to spriteNum + maxNodes

  repeat with i = 1 to 49*(insertPosition-1)
   --repeat with i = 1 to 49
   set the locH of sprite (firstNode+insertPosition-1) to ¬
```

```
      ((the locH of sprite (firstNode+insertPosition-1))+ 1)
  set the locH of sprite (maxNodes + firstNode+insertPosition-1) to¬      ((the locH of
sprite (maxNodes + firstNode+insertPosition-1))+ 1)
  updateStage
 end repeat

end moveRight
```

Insert Handler

```
on insertHandler
 set Token to field "Word"
 put "" into field "Word"
 Insert(Token)

end insertHandler
```

List Node Parent Script

```
-- Create new list node box
on new me, nodeNumber, cloneNode
 global firstNode,counter,listFlag,linkList

 set spriteNum to (nodeNumber + firstNode)

 set the castNum of sprite spriteNum to the number of cast "listNode"
 if count(linkList)=0 then
  set the locH of sprite spriteNum to -30
  set the locV of sprite spriteNum to 260
 else
  set the locH of sprite spriteNum to (40+49*(count(nodeList)))
  set the locV of sprite spriteNum to 260
 end if

 set counter to (counter+1)

 return me
end
```

List Value Parent Script

```
-- create a new list value slot
on new me, nodeNumber, cloneNode
 global maxNodes, firstNode,listFlag,linkList
 set j to nodeNumber + firstNode + maxNodes
```

```
 set valueNum to (nodeNumber + 38)
 set the castNum of sprite j to valueNum

 if count(linkList)=0 then
  set the locH of sprite (j) to -49
  set the locV of sprite (j) to 250
 else
  set the locH of sprite (j) to (21+49*(count(nodeList)))
  set the locV of sprite (j) to 250
 end if

end
```

List Delete Frame Script

```
on enterFrame
 global nodeList,listFlag,firstNode,maxNodes,newNode,counter,linkList
 global linkCount

 set listFlag to FALSE
 set linkCount to TRUE
 if count(linkList)=0 then
  alert "There is no node to delete!"
  beep
 end if

 set Token to field "Word"
 put "" into field "Word"
 Delete(Token)
end

on Delete Token
 global nodeList,maxNodes,deletePos,firstNode,linkList,tempList,counter
 global nodePos,linkCount

 set x to TRUE
 set sound to TRUE
 if Token <> ""  then
  set sound to FALSE
  repeat with i=1 to count(linkList)
   if getPropAt (linkList,i) = Token then
    set deletePos to i
    set x to FALSE
    --find the first node to delete and exit the repeat loop
```

```
    exit repeat
   end if
  end repeat

  if x=TRUE then
   alert"There is no such node in the list!"
   --go to frame "Insert"
   exit
  end if
 else
  if count(linkList)<>0 then
   alert "There is no input data to delete!"
   puppetSound "input"
   updateStage
   startTimer
   repeat while the timer < 6*30
    nothing
   end repeat

   exit
  end if
 end if

 if count(linkList) > 0 then
  --play sound
  puppetsound "delete"
  updateStage

  repeat with i=1 to count(tempList)
   if getPropAt (tempList,i) = Token then
    set nodePos to (i+1)
    set the text of member (firstNode+maxNodes+nodePos-3) to ""
    --set the nodes out the stage
    set the locH of sprite (firstNode+maxNodes+nodePos-2) to -20
    set the locV of sprite (firstNode+maxNodes+nodePos-2) to -10
    set the locH of sprite (firstNode+nodePos-2) to -20
    set the locV of sprite (firstNode+nodePos-2) to -10
    set counter to (counter-1)
    updateStage
    exit repeat
   end if
  end repeat
  repeat with i=nodePos to count(tempList)
   repeat with j = 1 to 49
    set the locH of sprite (firstNode+i-1) to ((the locH of sprite (firstNode+i-1))- 1)
```

```
      set the locH of sprite (firstNode+maxNodes+i-1) to ((the locH of sprite
(firstNode+maxNodes+i-1))- 1)
      updateStage
     end repeat
   end repeat
   startTimer
   repeat while the timer < 4*30
    nothing
   end repeat

  else
   alert "There is no node to delete!"

   exit
  end if

  if count(linkList)=maxNodes and Token="" then
   exit
  else
   deleteAt(linkList,deletePos)
   deleteAt(nodeList,deletePos)
   put (maxNodes-count(linkList)) into field "Que Nodes left"
   updateStage
   if count(nodeList)=0 then
    puppetsound "listemp"
    updateStage
    startTimer
    repeat while the timer < 7*30
     nothing
    end repeat
   end if
  end if

end Delete

on exitFrame
  go to "Insert"
end
```

"Help-Instruction" Frame Script

```
on enterFrame
  global firstNode,maxNodes,h,nodeList
  repeat with i = firstNode to (maxNodes + firstNode - 1)
   -- nodes 0 - 9 are sprites 29 - 38
```

```
  puppetSprite i, TRUE
 end repeat
 repeat with i = (maxNodes + firstNode) to (maxNodes * 2 + firstNode -1)
  -- values are sprites 39 to 48
  puppetSprite i, TRUE
  set the text of member (38 - maxNodes - firstNode + i) to ""
 end repeat

 repeat with j=(firstNode+maxNodes) to 48
  set h to the locH of sprite j
  set v to the locV of sprite j

  repeat with i=firstNode to (firstNode+maxNodes-1)
   set the visible of sprite i to FALSE
   set the locH of sprite (i+maxNodes) to (h+600)
   set the locV of sprite (i+maxNodes) to (v+600)

  end repeat
 end repeat

 updateStage
end

on exitFrame
 pause
end
```

"Help-About ADT" Frame Script

```
on enterFrame
 global firstNode,maxNodes,h,nodeList
 repeat with i = firstNode to (maxNodes + firstNode - 1)
  -- nodes 0 - 9 are sprites 29 - 38
  puppetSprite i, TRUE
 end repeat
 repeat with i = (maxNodes + firstNode) to (maxNodes * 2 + firstNode -1)
  -- values are sprites 39 to 48
  puppetSprite i, TRUE
  set the text of member (38 - maxNodes - firstNode + i) to ""
 end repeat

 repeat with j=(firstNode+maxNodes) to 48
  set h to the locH of sprite j
  set v to the locV of sprite j
```

```
    repeat with i=firstNode to (firstNode+maxNodes-1)
      set the visible of sprite i to FALSE
      set the locH of sprite (i+maxNodes) to (h+600)
      set the locV of sprite (i+maxNodes) to (v+600)
    end repeat
  end repeat

  updateStage
end

on exitFrame
  pause
end
```

2

VITA

Cong Xu

Candidate for the Degree of

Master of Science

Thesis: MULTIMEDIA VISUALIZATION OF ABSTRACT DATA TYPE

Major Field: Computer Science

Biographical:

Personal Data: Born in Qingdao, China, On August 3, 1970, the daughter of Mr. Jiashu Xu and Ms. Junrui Sun.

Education: Graduated from No. 9 High School, Qingdao, China in July 1988; Received Bachelor of Science degree in Chemistry from Ocean University of Qingdao, Qingdao, China in July 1992. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1997.

Professional Experience: Computer Assistant, Computing and Information Services, Oklahoma State University, August 1996 to December 1997; Teaching Assistant, Computer Science Department, Oklahoma State University, August 1997 to December 1997.

Professional Membership: The Association for Computing Machinery (ACM)