6-9-2014

TAIL: Data Structures Tutorial Site

Lauren Jauco
Santa Clara University

Ian Parker
Santa Clara University

Allie Rodriguez
Santa Clara University

Tyler Upadhyaya
Santa Clara University

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 9, 2014

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Lauren Jauco
Ian Parker
Allie Rodriguez
Tyler Upadhyaya

ENTITLED

TAIL: Data Structures Tutorial Site

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING
BACHELOR OF SCIENCE IN WEB DESIGN AND ENGINEERING

Thesis Advisor

Department Chair

TAIL: Data Structures Tutorial Site

by

Lauren Jauco
Ian Parker
Allie Rodriguez
Tyler Upadhyaya

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
Bachelor of Science in Web Design and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 9, 2014

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Any intermediate computer programmer will learn data structures. A data structure stores and organizes information into useful groups in order for a computer program to run as efficiently as possible. Data structures become extremely important when managing large amounts of data, whether it be as simple as managing statistics for a basketball team, or as critical as accounts for online banking.

The current way to learn data structures is through classes and online Java applets. In classes, most professors teach by drawing the data structure and editing the information on a board. Data structure tutorial websites mimic this visual way of teaching. No one, single site includes all types of data structures, which can be confusing and inconvenient for programmers. Current websites that teach data structures use Java applets which are obsolete, insecure, and run slowly. Most popular browsers disable Java by default because of these concerns. In order to use Java applets, users first find a site that has the proper data structure. Then, users must enable Java on the page in order to run the applet. When Java is disabled, browsers open a pop-up window informing users of Java's security risks and asking them if they would like to enable Java. Most users are discouraged by pop-ups because they are either unfamiliar with the warning or are concerned with security risks and will navigate away from the page. Finally, because each website is different, a user must adapt to the interface of that particular applet.

We will have built a website that provides users with one, single place to find all the data structures they need. Our website includes most of the common data structures and allows users to easily create their own if they are not included. We have developed a pseudocode language, and accompanying compiler, that allows users without advanced web programming backgrounds to create their own data structures. Users describe their data structures in a high level, pseudocode language which is then be translated into code appropriate for the web. Users are able to create data structures in our language and the compiler handles the programming necessary to display that data structure visually. Our site makes use of modern web technologies, such as

JavaScript and HTML5, to avoid the downfalls associated with Java applets. The website runs quickly and efficiently on all major Internet browsers to provide a hassle-free and hands-on tutorial for programmers wishing to learn data structures.

Chapter 2

Requirements Engineering

2.1 List of Requirements

We have considered many requirements when building our site in order to ensure that it functions properly. Our system must satisfy both functional and nonfunctional requirements to implement an effective product. Functional requirements refer to what the system must do, or tasks that it will perform, whereas non-functional requirements refer to the manner in which the tasks are carried out. Both are critical to consider while designing and building our data structures tutorial website in order for it to be a useful learning tool. Below is a list of the functional and non-functional requirements that we have defined for our project.

2.1.1 Functional Requirements

- The website will display animations
  - The website must be able to animate the data structures as they are constructed in order to demonstrate the process step by step to users
- The website will be interactive
  - Users must be able to add their own input values into our website to explore the different characteristics of various data structures
- The website will be error free
  - Since we are providing a learning tool, it is important that we do not convey (intentionally or unintentionally) inaccurate information
- The website will incorporate our new language
  - The website we develop must be compatible with our pseudocode language
- The language will be compatible with our website
  - Our pseudocode language must provide all the functionality needed to bring the website to life
  - This will allow users to use the language on their own website as well

2.1.2 Non-Functional Requirements

- The website will be intuitive
  - Users must be able to quickly learn the website
- The website will have a consistent look and feel
  - We want users to trust our website and must maintain a professional appearance
- The language must be simple
  - Our language will be both easy to read and easy to understand
- The language will be flexible
  - The language we create must be flexible and easy to manipulate

2.2 Use Cases

A use case is an example of the typical sequence of steps a user will perform when using our system. Our users consist of mostly of students learing data structures and developers using our language to create data structures.

2.2.1 Use Case Diagrams

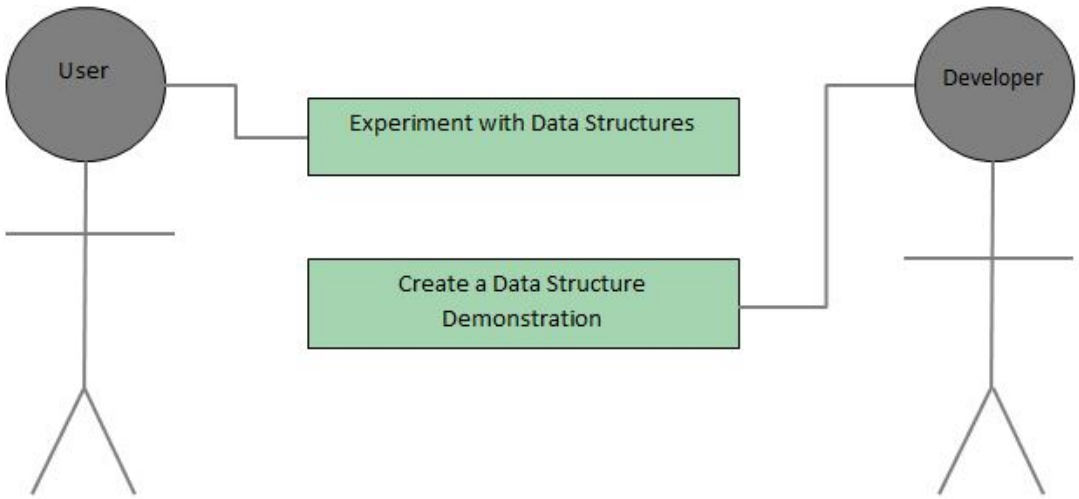

Figure 2.1: Use Case Diagram

Figure 2.1 demonstrates the general actions the actors will perform to use our website and language and which actors are associated with those actions.

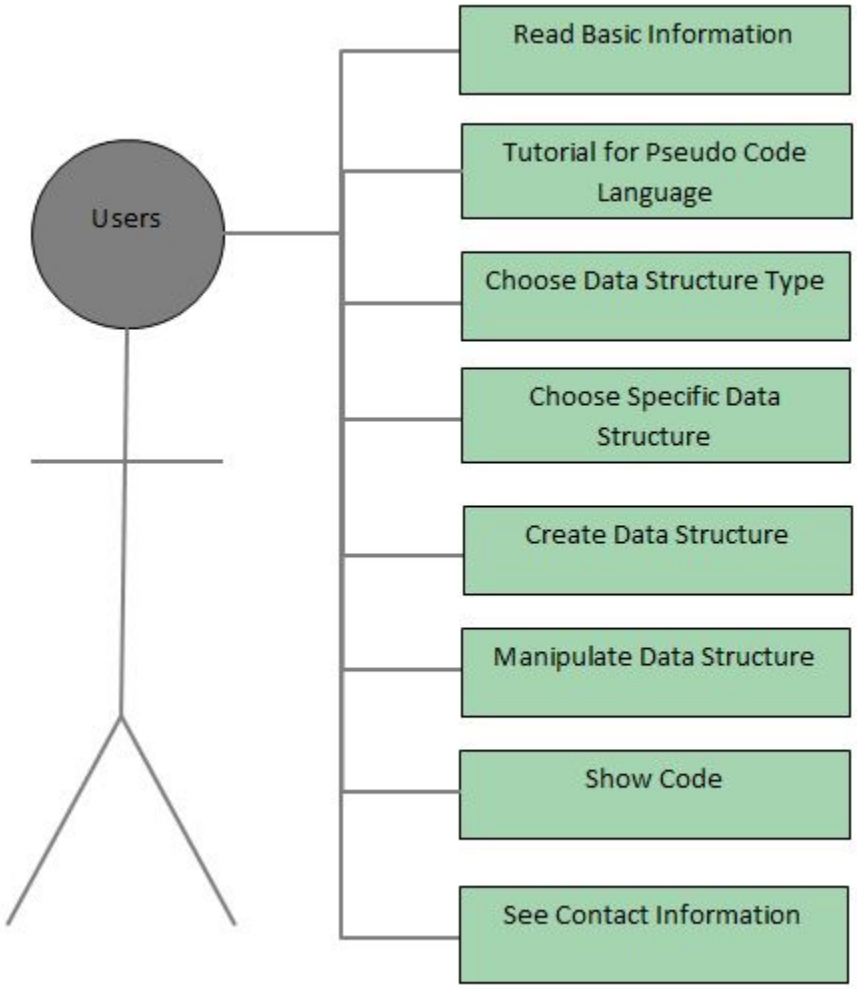

Figure 2.2: Detailed Use Cases for the Website

The diagram in figure 2.2 shows a more detailed layout of the specific actions a user may perform on our website.

2.2.2 Formal Use Case Specifications

Goal	Experimenting with data structures using our website
Actor(s)	Student
Preconditions	Access to a device with an internet connection
Postconditions	A deeper understanding of a particular data structure
Scenario	1. Navigate to our website on a device with an internet connection 2. Select the data structure they would like to work with from the icons on the landing page 3. Input any numbers of their choosing
Exception	1. Invalid input (a) If a user enters a value that is not supported (i.e. a letter), then an error message will be displayed: "Not a valid input value. Please try again." 2. Invalid Operation (a) If a user tries to delete a value that does not already exist in the data structure, then an error message will be displayed: "That value does not exist."

Table 2.1: Website Use Case

Goal	Using our language to create a data structure demonstration
Actor(s)	Developer
Preconditions	Access to a computer with an internet connection
Postconditions	JavaScript that can be run to visualize their code
Scenario	1. Navigate to our website on a computer with an internet connection 2. Download our compiler from our website 3. Compile pseudocode to output JavaScript that can be used to visualize their code
Exception	1. Code contains errors (a) Compiler will provide error documentation

Table 2.2: Language Use Case

Chapter 3

Design and Aesthetics

3.1 User Experience

Our data structures tutorial website must be intuitive in the way the program displays information and how the user interacts with its features. Clear graphics and animations are critical to capturing the professional look and feel that we are aiming for. The website's overall design direction is that of a 2D minimalism, using large blocks of color to divide pages with flat content and limited color diversity. The entire project, from the thesis, to the presentation, to the project itself, must look consistent. We have achieved this by using recurring aesthetic motifs and high resolution graphics and animations that work flawlessly on multiple platforms. The more simple the page style, the better it will scale to fit our goal platform resolutions. By working in Adobe Illustrator, we have created beautiful print and screen vector graphics that are, in themselves, far more scalable than raster graphics. In addition, these same ideas apply to all aspects of the animations. Animation speed, look, interaction, and capabilities must be consistent, interesting, but always functional. Complicated animation paths should not take away from how the object in use interacts with the rest of the page and how the viewer interprets its functionality. For instance, when building a binary tree the user should not witness both circles and squares as nodes for the sake of increasing aesthetic interest. We want users to enjoy using our website without working hard to visualize the data structures we will be demonstrating.

One way to emphasize the the projects unity and which materials are our own, is to include a logo on all web pages and project documents. Our logo contains certain color motifs that will re-occur throughout our designs. The use of limited color patterns has proven to be a successful marketing technique for companies like Drupal, Tiffany & Co., and Facebook. Our logo for TAIL: Data Structures Tutorial Website is a silhouette of a bunny with a color emphasis on its tail with a light green color. Website pages are grey scale except for the light green that will attract the user's eye to the most important elements and areas of the pages.

Color	Hex Value(0xRRGGBB)
Dark Grey Blocks	999999
Light Grey Background	F2F2F2
Near-black Logo	58595B
TAIL Green	A3D4AF

Table 3.1: Color Values

The logo was designed with the 2D minimalistic spectre in mind, focusing on simple shapes that combine to represent a larger picture, just as the visualization shapes of a data structure. The TAIL logo appears on every page in the top left corner along with a navigation bar (clicked pages are in light green) which orients the viewer within the website navigation map.

Figure 3.1: TAIL Logo

The home page in particular is the most important page because it is the landing page and the root connecting the user to all data structures. We have designed the site to be relatively shallow, focusing on the homepage as the primary navigation tool to access three different overarching types of data structures; arrays, trees, and linked lists. Paired with the description above, users should have a clear understanding of what our site does and how to begin using our program quickly and efficiently.

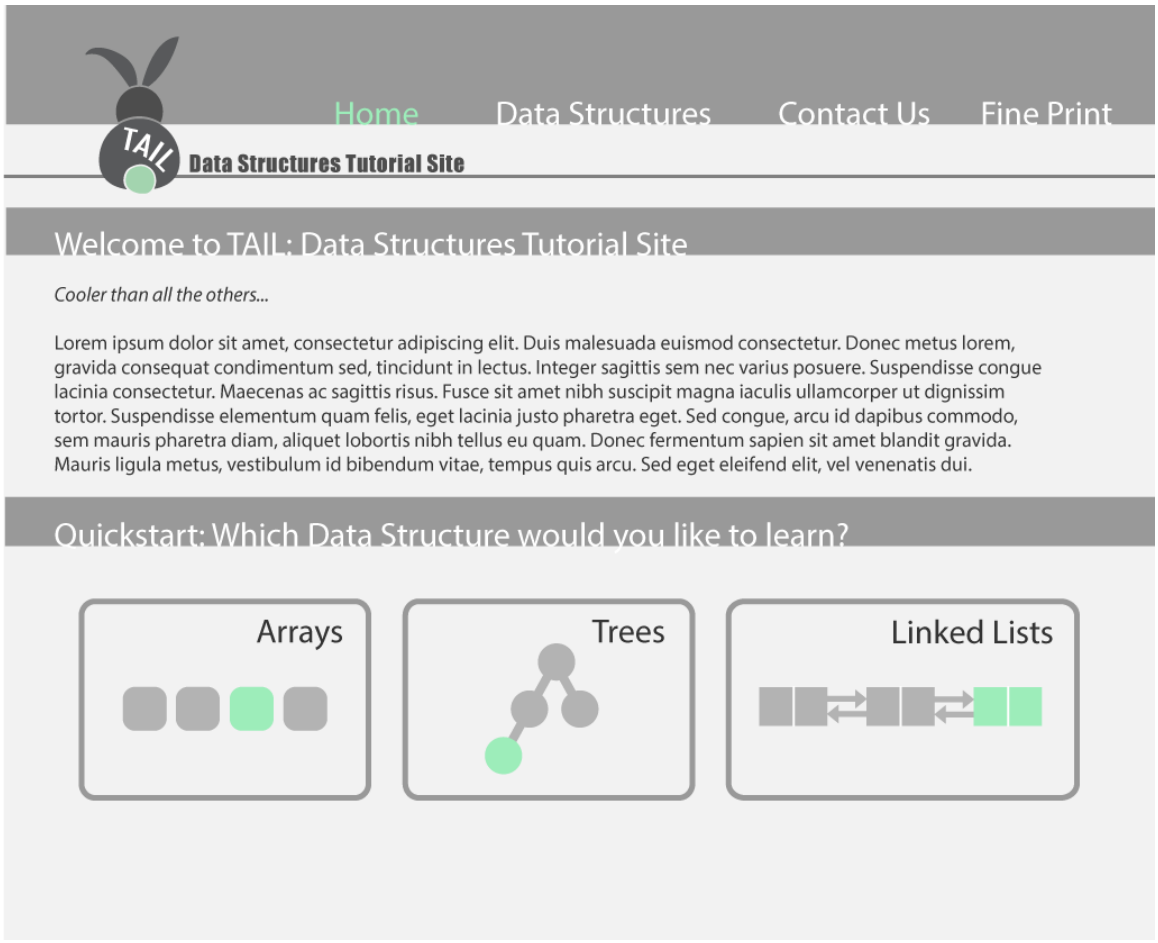

Figure 3.2: TAIL Home Page

Individual data structure pages all essentially contain the same functionality. All pages contain 3 main components:

1. The individual structure's quick description that is collapsible/expandable
2. An embedded visualization block for the structure created
3. A control panel connected to the right.

When the data structure is completed, users have the ability to click the large bottom right button S̈how Codeẗo view our site's generated code in a separate tab or window for that specific data structure. Users can then integrate this available code into their own program implementations.

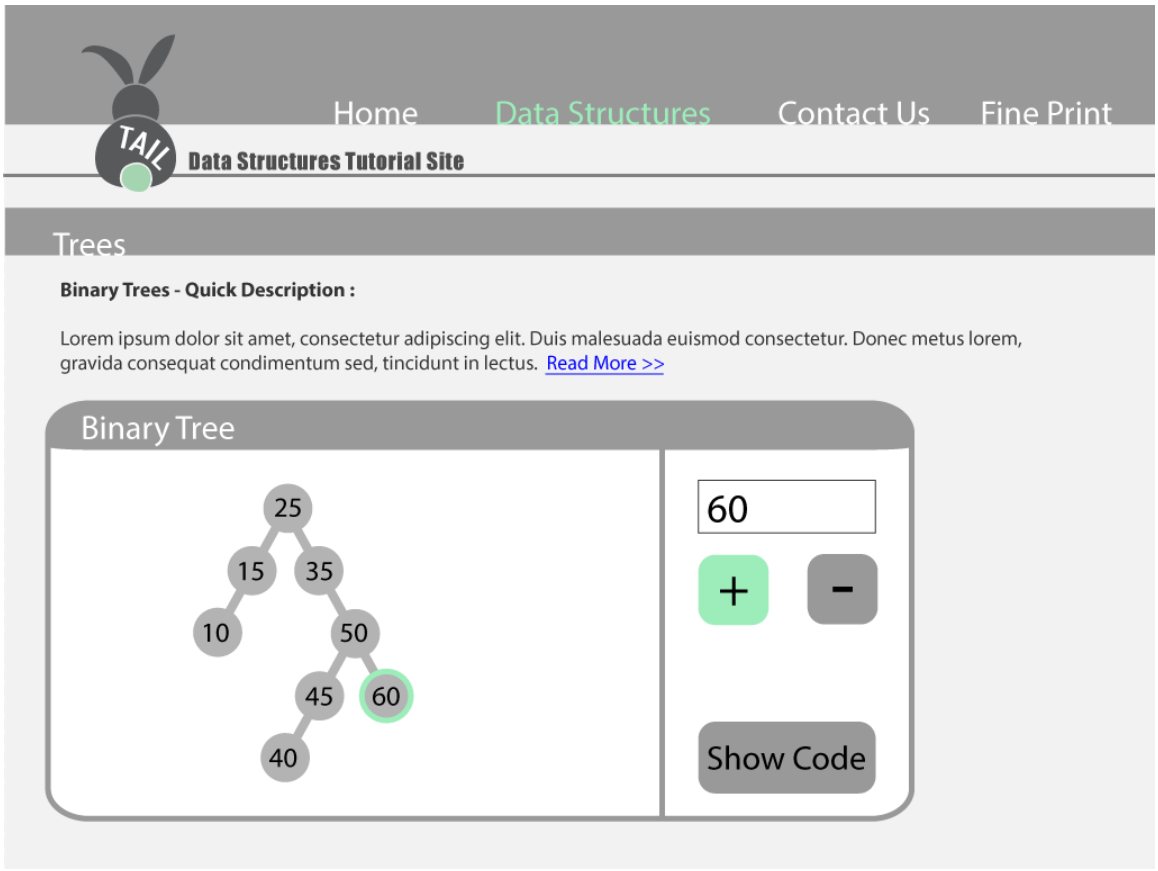

Figure 3.3: Sample Data Structure Page

Above all, the look of the site should support functionality by creating a comfortable learning environment. The look and feel of the website should never compromise usability considerations. We also must be incredibly careful with how the project looks on different platforms. We should not sacrifice aesthetic on an iPad to have an exceptional computer experience. Both experiences can be exceptional if the site is done minimalistically. Consistency and professionalism contribute to the overall trust from our users and encourages them to return to the site and share it with their peers. By considering the various media through which they could potentially view our site, we have demonstrated that we know who our customers are and have considered their needs when building the product.

3.1.1 Aesthetic Concerns Regarding the Audience

Although our target audience is students, older audiences could have aesthetic concerns such as the font being too small or the navigation not being clearly defined. Grey scale formatting can be attractive but must be sacrificed in the case of causing confusion for the viewer. Seeing that our website emphasizes the use of visuals to teach data structures, it would be important to limit our text, promote iconography, and emphasize our animations. This can be done by having text-focused pages and animation-focused pages separated. Too much information on one page could cause immense confusion for the user or, even worse, teach the material in an incorrect way.

3.1.2 Language Design

We have designed our language to be both easy to understand and read and easy to use and apply to a user's work. Many of the popular, general-purpose programming languages include features that complicate their syntax to the point where code can become almost unreadable. Our language is meant for only one specific task, writing data structures, so we are able to keep our syntax simple and focused. Programmers who are not familiar with our language should be able to read our code and immediately identify what the program does. Being easy to read will also allow our language to be easy to write. The uncomplicated syntax we developed makes our language simple to learn and allows programmers to master it quickly. The relatively small feature-set of our language also allows us to eliminate the need for many symbols in our code. Many languages use the less commonly used symbols to signify different actions within the code. Our language, because it does not support more complicated actions, does not require the use of these symbols. This will not only make the code more readable, but it will make it easier to type, as the only symbols needed are the ones easily accessible on a normal keyboard. Overall, our language has been designed primarily for simplicity, making it functional and aesthetically pleasing.

3.2 Design Rationale

We have been tasked with building a website that will serve as a visualization tool for students learning data structures. We considered using a number of different technologies to help us do so.

3.2.1 Website Design

Following our decision to design and implement a data structures tutorial website, we were faced with a crucial decision regarding front-end programming, whether or not to use a Javascript framework or a library. We were certain we were going to use Javascript for user interactions but were unsure of how we wanted to tackle the problem. At first, we were attracted to the framework functionality. Frameworks are powered by the user's most common needs in how it handles compilers, code libraries, and APIs (application programming interfaces). The framework, not the programmer, ideally removes gruelling hard coding by controlling the program's application directly and providing default cases. Frameworks, however, are supposed to be only extended upon rather than edited. If the framework is not behaving in the intended way, programmers are highly discouraged from editing the framework's code due to the risk of ruining the framework and in turn breaking their own program. We looked at two specific JavaScript frameworks, SproutCore and Ember.js. Both are open-source client-side frameworks, with Ember.js previously being named SproutCore 2.0. Although very similar, Ember.js is a more lightweight version of SproutCore but still provides the same large user interface framework with a specific and sometimes limiting toolchain and workflow. Users can near complete a project using a framework only to find that they need to re-write it due to a complication with specific limitations that frameworks place on particular functionality. We found that a framework would be too complicated for our project and began focusing on JavaScript libraries.

During our research we were referred to Data-Driven Documents, or D3.JS, a Javascript library focused on data control and influence to create dynamic and interactive graphics and animations. Known for clean and inclusive data visualization, we found that this system would be perfect for creating and displaying data structures. D3.JS utilizes the combination of SVG (Scalable Vector Graphics), JavaScript, HTML5, and CSS3 (Cascading Style Sheets) and is known for giving its users great control over the final visual product. Despite its relatively recent creation, D3.JS has a useful support webpage (http://d3js.org/) that contains a multitude of examples, tutorials, and documentation. Installing and using D3.JS was easy and quick to learn. Nearly all reviews of D3.JS have been positive, speaking of the succinct syntax and its similarity to jQuery (a library we all have experience using). The library contains pre-built JavaScript functions to select elements, create objects, style/transition/effects. By using a JavaScript library, we will have more control over the content creation and data interaction resulting in a personalized visual product. We will use lightweight

D3.JS programming to design the animated data structures and buttons used by our viewers to learn data structures in a fluid yet cohesive way.

3.2.2 Language/Compiler Design

We chose to create our own pseudocode language in order to increase understanding and usability of the JavaScript and D3JS libraries. While JavaScript may be easy to use for the experienced programmer, newer programmers tend to struggle with it. A language that simplifies the amount of programming while still allowing for advanced use of the libraries helps new programmers learn the full capabilities of programming in a faster and more efficient way. The straight forward nature of pseudocode allows for users to easily create new data structures and sorting algorithms.

Writing a language does have its drawbacks. Defining a language that can handle everything that we need it to do can be extremely difficult. It also means that we have to create an accompanying compiler in order to produce the appropriate JavaScript. Although making a language does have some disadvantages, we believe that the usability and abstract nature of our language provides many advantages.

We chose to use C++ to create our compiler. C++ allows the programmer to utilize the powerful, low-level features of C within an object oriented environment. This object oriented environment provides us with many useful tools for managing the relationships between data that we would not have access to in C. Aditionally, we already have some familiarity working with C++, and will not have to waste valuable time learning a new language.

Chapter 4

Development

4.1 Technologies Used

During the development of our website, pseudocode language, and accompanying compiler, we utilized various forms of technology.

Google Drive

This is used for file storage and collaboration. When one of our team members cannot be present to work on documentation, Google Drive allows us to work simultaneously and remotely. It also provides a secure place for us to keep all of our research and documentation without the need to worry about who has the most up to date information. Instead, everything is shared and updated instantaneously.

LaTeX

This is a document markup language that we are using to construct our final documentation.We will utilize convenient features such as the numbering and labelling tables and figures, construction of charts, and overall formatting. We found it beneficial to take the time to figure out how to use LaTeX because although it is not entirely intuitive, it will save us time with formatting later in the process.

D3.js

This JavaScript library is used to create basic animations on our website. D3.js includes functionality for grouping data within a Document Object Model (DOM) and then manipulating it visually on the page. The documentation for D3.js stresses the efficiency and lightweight nature of the library.

C++

Our compiler will be implemented in C++. C++ combines the low-level power of C with the utility provided by an object oriented language. It also allows us to easily deploy our compiler on Mac and Linux and gives us the ability to deploy on Windows with minimal extra effort. We also have prior experience with C++ which will allow us to begin coding without having to spend time learning a new language.

Subversion and Git

As with any code-based product, version control is a vital part of our process. As we all work on different aspects of our website independently, we need a way to eventually combine our changes to the code, have access to the changes made by others, and later reference changes we made on a certain date or version from past updates. We attempted to use both Subversion and Git as our version control service. Overall, the hassle of setting up and using either of these systems proved to be greater than the benefit we gained from them and we decided to manage our version control by hand.

4.2 Risk Analysis

Specific probable and severe risks involved in the development of the application are listed in the table below alongside the mitigation strategies our team planned to employ to ensure the success of the project application.

4.2.1 Risk Analysis Tables

P = Probability, S = Severity, I (Impact) = P · S

Risk	Consequence	P	S	I	Mitigation
Time	Not finishing	0.50	6	3.0	• Prioritize features for completion. Features which are essential for basic operation have highest priority • Utilize Gantt chart to stay on schedule and identify if any additional resources can be reassigned • Allow for team members to assist in other areas if needed

Risk	Consequence	P	S	I	Mitigation
Unfamiliar Technology	Learn new Technologies	0.50	4	2.0	• Plan time to learn and use the new technologies

Risk	Consequence	P	S	I	Mitigation
Language Complexity	Have to redo the language specification	0.20	8	1.6	• Have users program in the language to see if it is easy to program in

Risk	Consequence	P	S	I	Mitigation
Illness	A group member falls behind on work	0.50	3	1.5	• Divide work among other group members to catch up • Group members will work closely with each other to create redundancy of knowledge and information

Risk	Consequence	P	S	I	Mitigation
Website Acceptability	Have to redo the website layout or color scheme	0.20	6	1.2	• Conduct user surveys and acceptance testing

Risk	Consequence	P	S	I	Mitigation
Loss of Code	Starting over	0.05	8	0.4	• Use version control • Keep local backups

4.3 Development Timeline

In order to stay on track with our design, implementation, deliverables, testing, and presenting, we developed a timeline in the form of a Gantt chart. The first chart includes all of the tasks that we expected to perform so that we could ensure that we completed each component on time. The second chart shows an updated Gantt chart detailing how each part of our prject was actually completed along with a sample of the timeline for the TAIL Four Step Process.

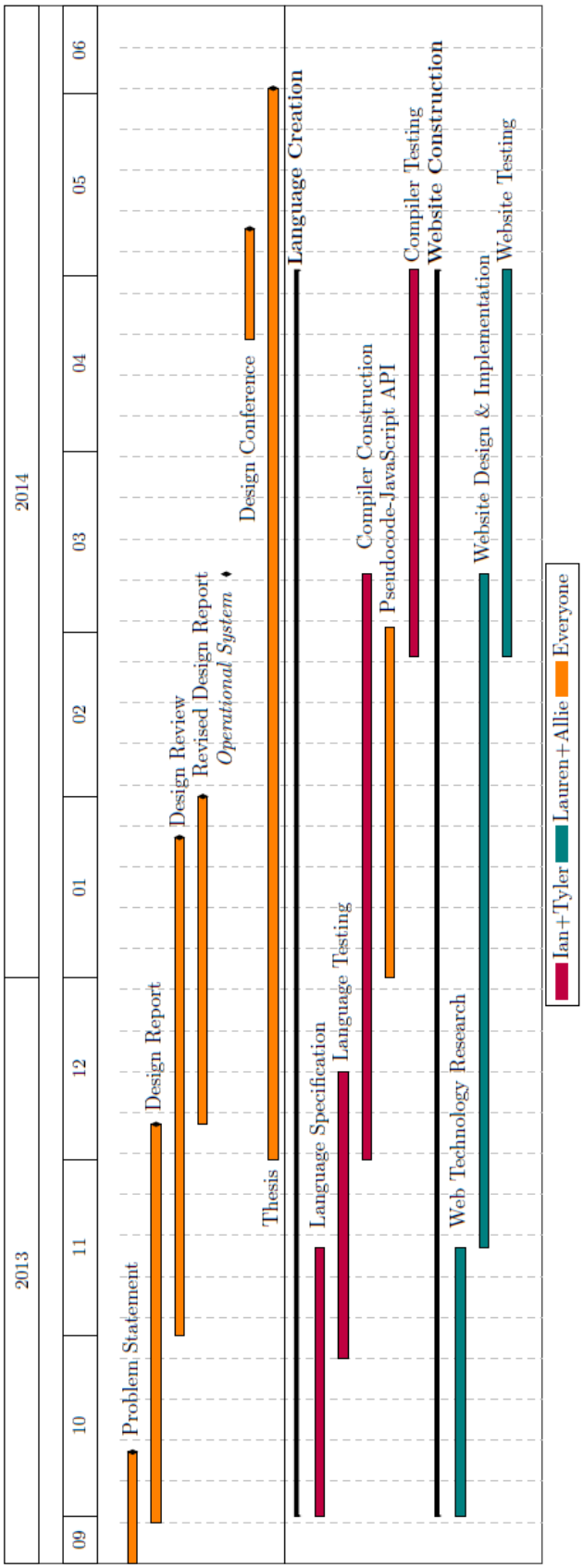

Figure 4.1: Gantt Chart (original)

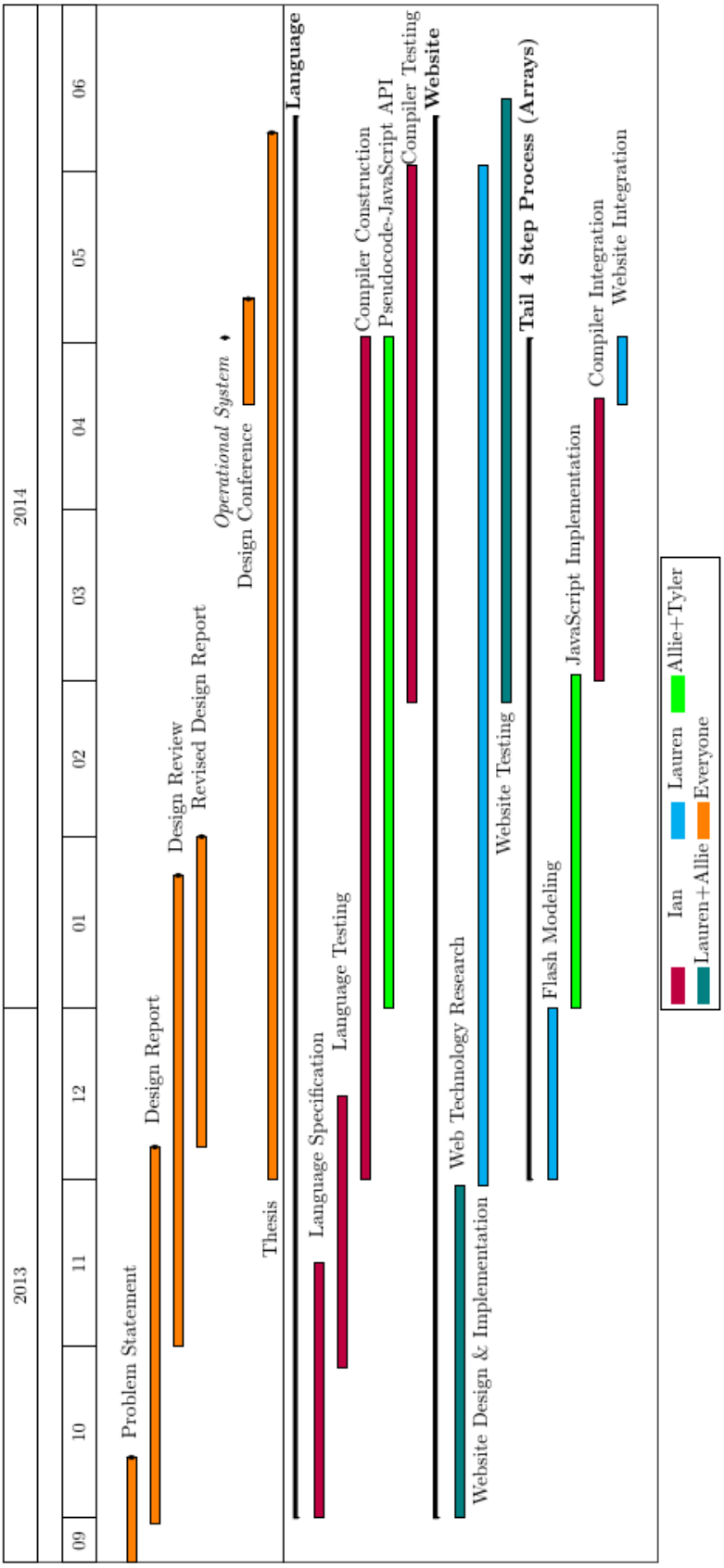

Figure 4.2: Gantt Chart (updated)

4.4 Test Plan

To ensure that our language, compiler, and website all work together harmoniously, we developed a plan to perform a series of tests on our system. Listed in the following subsections are the types of tests we have completed or plan to enact on our system.

- **Language**
  - Functional Testing
    - Sample programs were written in the language to ensure that it is capable of describing data structures of all types
    - Both simple and complex data structures were tested
- **Compiler**
  - Unit Testing
    - Individual components of the compiler were tested as they are constructed
    - Each developer was responsible for unit testing code he develops
  - Functional Testing
    - Once the compiler is completed, sample programs were run through the compiler
    - Test cases consisted of both legal and illegal programs to make sure the compiler works properly and catches errors
- **Website**
  - Usability Testing
    - The website was tested to make sure the user interface is both easy-to-use and uncomplicated
    - The website was evaluated along 4 parameters: efficiency, accuracy, recall, and emotional response
  - Acceptance Testing
    - We will enlist the assistance of various undergraduate computer engineering and computer science students to give us feedback on our website's usability
    - Our testers will be asked to measure aesthetics, ease of use, and helpfulness

Chapter 5

Language

5.1 Grammar

The following constitutes the grammar for TAIL in Backus-Naur Form:

⟨*structure*⟩ → ⟨*type*⟩ **new id begin** ⟨*init-block*⟩ ⟨*functions*⟩ **end**

⟨*type*⟩ → List
| Tree
| ArraySort
| ListSort
| Array

⟨*init-block*⟩ → **init begin** ⟨*initilizations*⟩ **end**
| ⟨*empty*⟩

⟨*initializations*⟩ → ⟨*initialization*⟩ ⟨*initializations*⟩
| ⟨*empty*⟩

⟨*initialization*⟩ → **init-property** : **init-value**

⟨*functions*⟩ → ⟨*function*⟩ ⟨*functions*⟩
| ⟨*empty*⟩

⟨*function*⟩ → **id** (⟨*parameters*⟩) ⟨*return-type*⟩ **begin** ⟨*statements*⟩ **end**

⟨*parameters*⟩ → ⟨*parameter-list*⟩
| ⟨*empty*⟩

⟨*parameter-list*⟩ → **id**
| **id** , ⟨*parameter-list*⟩

⟨*return-type*⟩ → **returns return-value**
| ⟨*empty*⟩

⟨*statements*⟩ → ⟨*statement*⟩ ⟨*statements*⟩
| ⟨*empty*⟩

⟨*statement*⟩ → **begin** ⟨*statements*⟩ **end**
| **return** ⟨*expression*⟩
| **while** ⟨*expression*⟩ **begin** ⟨*statement*⟩ **end**

| **for** ⟨*expression*⟩ **to** ⟨*expression*⟩ **by** ⟨*expression*⟩ **begin** ⟨*statement*⟩ **end**
| **if** ⟨*expression*⟩ **then** ⟨*statement*⟩ **else** ⟨*statement*⟩
| **if** ⟨*expression*⟩ **then** ⟨*statement*⟩
| ⟨*expression*⟩

⟨*expression*⟩ → ⟨*expression*⟩ := ⟨*expression*⟩
| ⟨*expression*⟩ || ⟨*expression*⟩
| ⟨*expression*⟩ && ⟨*expression*⟩
| ⟨*expression*⟩ = ⟨*expression*⟩
| ⟨*expression*⟩ != ⟨*expression*⟩
| ⟨*expression*⟩ <= ⟨*expression*⟩
| ⟨*expression*⟩ >= ⟨*expression*⟩
| ⟨*expression*⟩ < ⟨*expression*⟩
| ⟨*expression*⟩ > ⟨*expression*⟩
| ⟨*expression*⟩ + ⟨*expression*⟩
| ⟨*expression*⟩ - ⟨*expression*⟩
| ⟨*expression*⟩ * ⟨*expression*⟩
| ⟨*expression*⟩ / ⟨*expression*⟩
| ⟨*expression*⟩ % ⟨*expression*⟩
| ! ⟨*expression*⟩
| - ⟨*expression*⟩
| ⟨*expression*⟩ [⟨*expression*⟩]
| **id** (⟨*expression-list*⟩)
| **id** ( )
| **num**
| (⟨*expression*⟩)
| **id** . ⟨*members*⟩

⟨*expression-list*⟩ → ⟨*expression*⟩
| ⟨*expression*⟩ , ⟨*expression-list*⟩

⟨*members*⟩ → **id**
| **id** . ⟨*members*⟩
| **id** . ( )
| **id** . (⟨*expression*⟩)

5.1.1 Notes

The options under the ⟨*type*⟩ section are the built-in supertypes that users may base their data structures on. List, Tree, ArraySort, and ListSort are the most basic building blocks for data structures and sorting algorithms and should serve as a foundation sufficient for the modeling of more advanced data structures.

The **init-property** and **init-value** are initialization options specific to each basic ⟨*type*⟩. They will provide the user with customization options for their data structure.

5.2 Code Sample

The following is a sample implementation of selection sort written in TAIL.

```
ArraySort new selectionSort begin

init begin
        arrayOrder: random
end

ssort() begin
        for i := 0 to i < length(selectionSort)-1 by i := i + 1 begin
                min := i;
                #moveToPos i;
                #putInBox selectionSort[i];
                for j := i+1 to j < length(selectionSort) by j := j + 1 begin
                        #moveToPos j;
                        if selectionSort[j] < selectionSort[min] then begin
                                #putInBox selectionSort[j];
                                min := j;
                        end
                end
                if min != i then begin
                        swap(i, min);
                end
        end
end

end
```

Chapter 6

Societal Issues

6.1 Ethical

Our project is first and foremost a teaching tool. Our users will trust us to create a product that is free of errors. Our website contains examples of various data structures that our users will interact with and learn from. It is our responsibility to make sure that all of these examples are accurate and function exactly as expected. We also have a responsibility to our users to ensure their security while they are on our website. We do not store any of our users personal information, but we must still make sure our website is free of any malicious content placed there without our knowledge. Our website allows users to download various files related to our compiler and we must make sure that these downloads do not contain any malicious software that may infect our users computers.

6.2 Social

Our project is intended to provide a service to computer science students of all ages. We provide this service free of charge to anyone who comes to our website. We developed our system, not as a source of profit, but to help any students who may come after us. In creating it, we wished simply to replace outdated tools so that future students have a better experience than we did when we were learning.

6.3 Political

There are no political concerns with our project. We intend only to create a simple teaching tool to replace the existing, outdated tools currently available.

6.4 Economic

Currently we have no plans to try to generate any income with our project. However, if we were to, we could charge an initial fee in order to view any of the data structures that we have available. We also have room to add advertisements to our website. However, we believe that this would take away from the main point of this website. We wanted users to be able to come our site and just focus on the learning. This is why we designed to the language and the website to be the way it is. Also, users do not need to look at how data structures work every day and therefore would not want to pay for a site that they may only use a couple of times.

6.5 Health and Safety

Our project has no health and safety concerns. It is a website, language, and compiler. Therefore, health and safety is not a concern for this project.

6.6 Manufacturability

Our project is very manufacturable, do to way we designed our language. We made it very simple for people to create their own data structures, and would allow the user to put their data structures on our sight. Therefore, along with our team, we would be able to increase the number of data structures available very quickly. As far as space, all you would need is to host the website and a place to store all of the uploaded data structures. Therefore, our project is extremely manufacturable.

6.7 Sustainability

In a narrow sense, our project is sustainable in that it is written in the most modern programming languages of our time. As JavaScript, HTML and CSS continue to evolve and grow in the future, they will remain compatible with our website for much longer than if we had written them in languages that go through more fundamental changes as they are updated. Unlike Java applets, users of TAIL will not need to update any of their software to be able to use our website or compiler. While our technology continues to move towards the mobile sector, TAIL is sustainable in that it does not require any additional software to be installed besides a browser, which has already been adapted to mobile devices. As far into the future that browsers are compatible with Javascript, HTML and CSS, our site will remain fully functional and relevant.

In the broader sense, TAIL is a sustainable system in that it efficiently utilizes resources that are currently available. As opposed to textbooks and other educational documentation, our site contains everything a user needs to know to gain an understanding of data structures using the smallest amount of resources necessary.

All one needs is a computer, tablet, or smartphone which the majority of students will have anyway for other academic endeavors. By using the web, we have limited the amount of physical tools necessary to learn about data structures and therefore created a sustainable learning tool that students will be able to utilize for years to come.

6.8 Environmental Impact

TAIL has minimal impact on the environment. Arguably the server on which it is hosted consumes a number of physical tools however we felt that by making our tool available online we were utilizing as few resources as possible and therefore having the smallest amount of impact on the environment around us.

6.9 Usability

Being an educational tool, it was imperative that TAIL be as user-friendly as possible. We spent months carefully designing the web map, layouts, animations, colors, images, and language syntax in order to achieve a seamless user experience. By implementing a 2-D minimalistic design we ensured that our site would render correctly not only on a desktop computer browser, but mobile browsers as well. Our goal was to create a familiar, aesthetically pleasing environment in which students could experiment with data structures.

6.10 Lifelong Learning

Being a tutorial learning website, the TAIL projects objective is to encourage lifelong learning in its users. Data Structures is a fundamental programming topic leaving TAIL with a moral responsibility to provide accurate information and reliable documentation of both our product and resources. The ability to program advanced data structures is a powerful tool, especially in the modern digital age. Programmers, as we all know, can use their skill set in immensely detrimental ways.

6.11 Compassion

We released a disclaimer describing the responsibility of the site as well as the responsibility of the programmer visiting our site. We need our site to remain a learning tool, rather than a mean to develop damaging programs. We built TAIL: Data Structures Tutorial site to fix a problem that we personally experienced and hope that the site helps future novice programmers visually learn data structures.

Chapter 7

Lessons Learned

In the fall, we designed an initial Gantt chart to map out the following six months of work. The Gantt chart appropriated product work and time per person leading up to the Senior Design Conference in early May. The original graph was an ideal case and was not completely followed, which, in reality, it should not have been. The first critical lesson we learned as a group was the necessity of developing a process. The goal of our project seemed daunting as of November, but after analyzing and mapping out the abilities of our members taking into account our resources, we realized that we had developed our own TAIL Four Step Process. The TAIL team consists of four members serving a variety of integrated tasks; a web designer, a compiler expert, and two JavaScript animators. We designed, created, and implemented each data structure type using this TAIL Four Step Process.

The TAIL Four Step Process

1. The web developer designs high-level flash animations to serve as a guide for the end product.
2. The JavaScript team builds and animates the data structure making it interactive.
3. The compiler expert constructs a compiler that is able parse our pseudocode language and output JavaScript objects and functions that are needed to display and manipulate the data structures.
4. The web developer takes the finished data structure animations and integrates it into the website.

Despite developing a rather seamless process, we had our own time limitations for the project. The process is time consuming, especially the first time around, and we needed to optimize our productivity. Our second major lesson was learning how to work in parallel rather than in sequence. The members of TAIL had well identified, unique roles where each member was crucial to continuing a data structure down the TAIL Four Step Process pipeline. However we learned that if one member was unable to meet his or her work goal for the week, it quickly set us back on our timeline. In February, our web developer fell extremely ill, and

production came to a standstill. To complete a project that relies heavily on sequential steps, members need to be able to share responsibilities especially during extraneous cases.

The final major lesson we learned was the importance of taking time to make seemingly preliminary design decisions and being realistic about the outcome. Taking extra care to research all available options saved us a lot of time in the long run. In particular, we researched multiple JavaScript libraries and frameworks to decide which management system would best serve our needs, and ultimately which specific program we would use. Research took nearly three weeks. Eventually we agreed on the D3.JS library after being initially attracted to the capabilities of frameworks. Had we jumped into whatever library/framework, we would have most likely wasted time and resources on a system that did not best fit our project requirements.

The Gantt chart mapped out the timeline to match all the ideal requirements we initially set in October. We hoped to implement majority of algorithms for arrays, trees, and some linked lists before the design conference. Based on our projects process and mistakes, we needed to be realistic about what we could and could not finish. The Gantt chart has now been updated to reflect the timeline that occurred rather than the ambitious one set in October.

Chapter 8

Future Work

8.1 Usability Testing

In the near future we plan to conduct usability studies on all aspects of TAIL: the website interface, pseudocode language, and compiler.

8.2 Tweaks to Animations

Due to time constraints, the animations displayed on our website are the ones easiest to implement using JavaScript that contains the building block animations that can easily assemble future data structures types. The current data structures jump instead of move and this may not be the optimal way to display the data structure transitions. Therefore we plan to implement a variety of transition techniques and test them against programming students to find the animation with the highest convergence. We hope to make data-backed decisions, rather than easy fixes, to display the structures on the screen.

8.3 More Features for Trees

As of right now a user can add to, delete from, and search a binary search tree. We would like them to be able to traverse a tree as well. Pre-order, post-order, and in-order traversals will be the next feature applied to trees on the TAIL website. This will be displayed as a radial button option adjacent to the current action box.

8.4 Different Types of Trees

Currently we have only implemented binary search trees on the TAIL website. In the future we hope to include other types of trees such as binary, AVL, and B trees. Our goal is to include implementation of various tree types on our website website, complete with their definitions to be as complete as possible.

8.5 Add Linked Lists to the Site

Following the completion of arrays and trees, the next type of data structure we aim to cover is linked lists. We would like to provide users with a few examples of linked list algorithms so they have examples that they can build upon using the TAIL pseudocode language. We consider linked lists one of the fundamental data structures for an aspiring programmer to learn therefore we feel it important to include at least the simplest instance of linked lists on our TAIL website for the sake of completion. We are at Step One of the TAIL Four Step Process, with high-level flash animations of singly-linked lists and doubly-linked lists.

8.6 Continued Improvement on the Language Syntax

Before completion of this project, we plan on making small improvements to the syntax of the TAIL pseudocode language. As we continue to develop new algorithms, we may find that slight adjustments to the syntax of the language are necessary for straightforward and accurate development. Additionally, after performing usability studies on our language, we anticipate users to have feedback on the ease of use of the TAIL language and we plan to make changes accordingly.

Chapter 9

Appendix: User Manual

9.1 TAIL Website User Manual

We designed the website to provide a smooth user experience. When you first land on our home page, you are offered five choices on the navigation bar. The Home and The Project tabs are fairly self-explanatory. Home will always take you back to the home page with the image slider. On The Project page, viewers can read the Santa Clara University Senior Design Conference description and learn more about the TAIL project.

The Structures tab navigates you to the first offered data structure, which currently is minimum value array. On the left, you will find a list of all of our offered data structures grouped by three overarching categories: Arrays, Trees, and Linked Lists. After choosing your desired data structure, it appears in the adjacent action box along with a definition and BigO notation. Depending on the data structure, users can perform a variety of functions. For instance, when you enter a value into the search bar, you view the visual representation of the particular data structures searching algorithm. Functions like sort, insert, delete, and reset (which randomly generates a new array for variety) are also provided, with each responding appropriately given the type of data structure demonstration. All animations can be paused and resumed. This is incredibly useful for learners to follow along with the animation at their own pace.

The following two diagrams give examples of the TAIL site and how the user can interact with them.

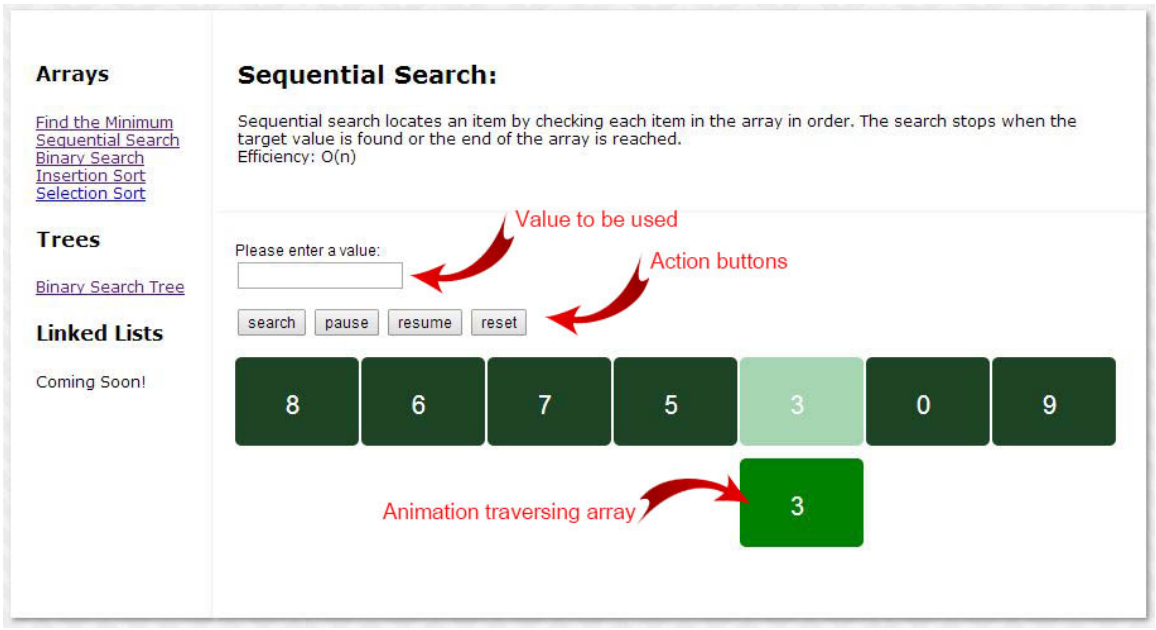

Figure 9.1: Array Functionality

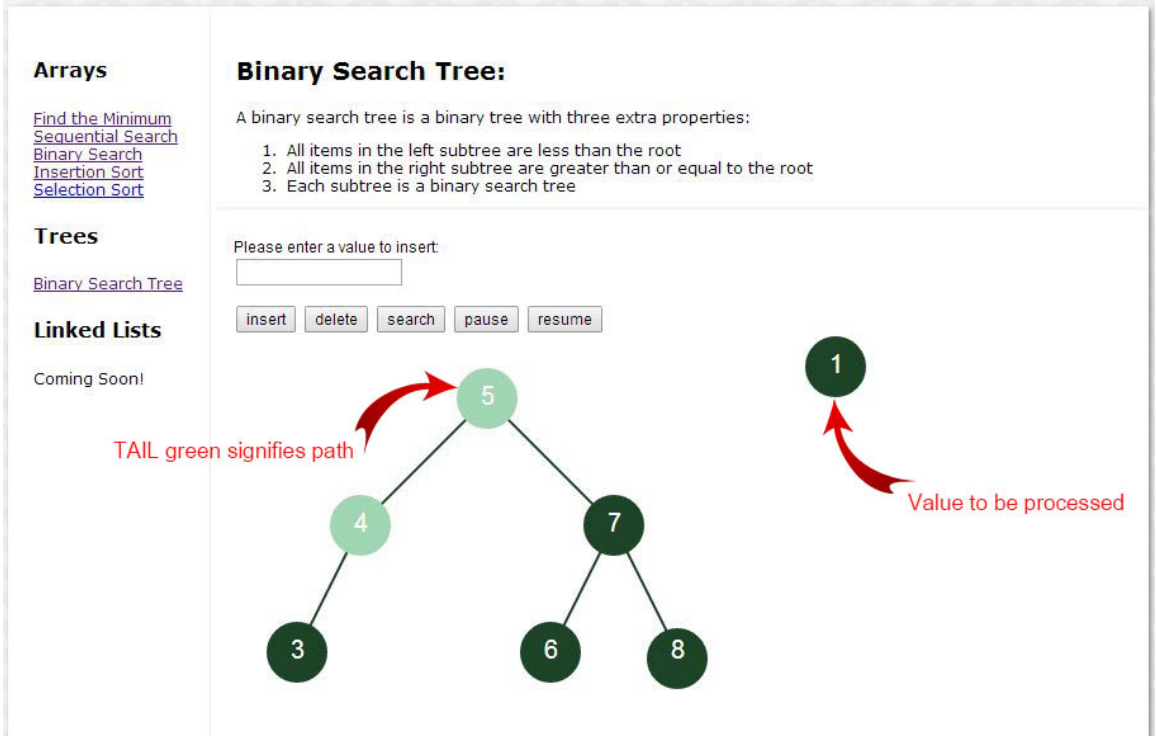

Figure 9.2: Tree Functionality

9.2 TAIL Language Reference

9.2.1 Tutorial

This tutorial will guide you through the creation of a simple data structure in order to familiarize you with the TAIL language. For the purposes of the tutorial, the array sort algorithm selection sort will be demonstrated.

Every TAIL program begins with a declaration. This declaration informs the compiler which type of algorithm you are creating so it can load the proper TAIL utilities and animations for you.

```
ArraySort new selectionSort begin
. . . implementation goes here
end
```

The declaration has 3 components. The first, in our case ArraySort is the type of the algorithm. The second is the new keyword, this tells the compiler that this is a declaration of a new algorithm. The third part is the name you have chosen for your algorithm, in our case we have chosen selectionSort. This name will also serve as a reference to the data structure we are working with. Following the declaration is the beginning of a block which will contain the code for our algorithm.

Next, the program has an optional initialization block

```
init begin
        arrayOrder: random
end
```

This block contains one or more of the initialization options for the data structure you are working with. In our case we have chosen to initialize our ArraySort with random values.

Finally, the program may contain one or more functions. These functions are where you define the algorithm of your data structure. They will be translated into the appropriate JavaScript code for you to use in your website or other JavaScript code. Their names will be kept the same as the name you choose for them.

```
ssort() begin
    for i := 0 to length(selectionSort)-1 by 1 begin
        min := i;
            #moveToPos i;
            #putInBox selectionSort[i];
            for j := i+1 to j < length(selectionSort) by j := j + 1 begin
                #moveToPos j;
                if selectionSort[j] < selectionSort[min] then begin
                    #putInBox selectionSort[j];
                    min := j;
                end
            end
            if min != i then begin
            swap(i, min);
            end
    end
end
```

Our program contains constructs which should be familiar to you from your other programming experience, such as if statements and for loops. They may be a little different than they appear in your favorite programming language but they function in the way you would expect. Our program also contains statements prefixed by a # symbol. These are special animation function calls which are unique to TAIL. A full reference on the available animations is available at the end of this tutorial.

Thats it. TAIL was designed to be simple to learn. Hopefully, your previous programming experience will allow you to quickly learn TAIL and program your own data structures.

9.2.2 Language Utilities

Types

```
Tree
List
Array
ArraySort
ListSort
```

Arrays

`swap(index1, index2)`
Swaps the values at index1 and index2 within the array

`length(array)`
Returns the length of array

Lists

`next(a_node)`
Returns the node following a_node in the list

`previous(a_node)`
Returns the node before a_node in the list. Only works on doubly linked lists.

`head`
The head pointer of the list. Only works if the list is set to have a head pointer.

`tail`
The tail pointer of the list. Only works if the list is set to have a tail pointer.

Trees

`child(node, child_index)`
Returns nodes child at child_index

`left(node)`
Returns the left child of a binary tree node. This function is equivalent to child(node, 0)

`right(node)`
Returns the right child of a binary tree node. This function is equivalent to child(node, 1)

9.2.3 Initilization Options

Arrays
`arrayOrder:` `random, increasing, decreasing`
- Initializes the array with values in random, increasing, or decreasing order.

Lists
`linking:` `single, double`
- Creates a list as either singly or doubly linked. Default is single.
`headPointer:` `yes,no`
- Sets up a head pointer for the list. Default is no.
`tailPointer:` `yes,no`
- Sets up a tail pointer for the list. Default is no.

9.2.4 Animation API

Animation commands are prefixed by the # symbol. They control the way in which the data structures are displayed on the screen.

Arrays
`putInBox number`
Puts number in the box

`moveToPos index`
Moves the box below the given index

Lists
Available in a future version

Trees
Available in a future version