# Laboratory 9A

# Abstract Data Types

## Objective

- Gain a deeper appreciation for stacks, queues, and trees.

## References

*Software needed:*

1) A web browser (Internet Explorer or Netscape)
2) Applets
   a) Stackqueue
   b) Trees

*Textbook reference:* Chapter 9, pp. 286–297

# Background

There are many abstract data types in use in Computer Science. Chapter 9 of your textbook covers those most commonly used. In this lab, you will watch stacks, queues, and trees operate in order to gain a deeper understanding of what they look like and how they function. In the next lab, you will investigate how searching and sorting are influenced by the data types used to store values.

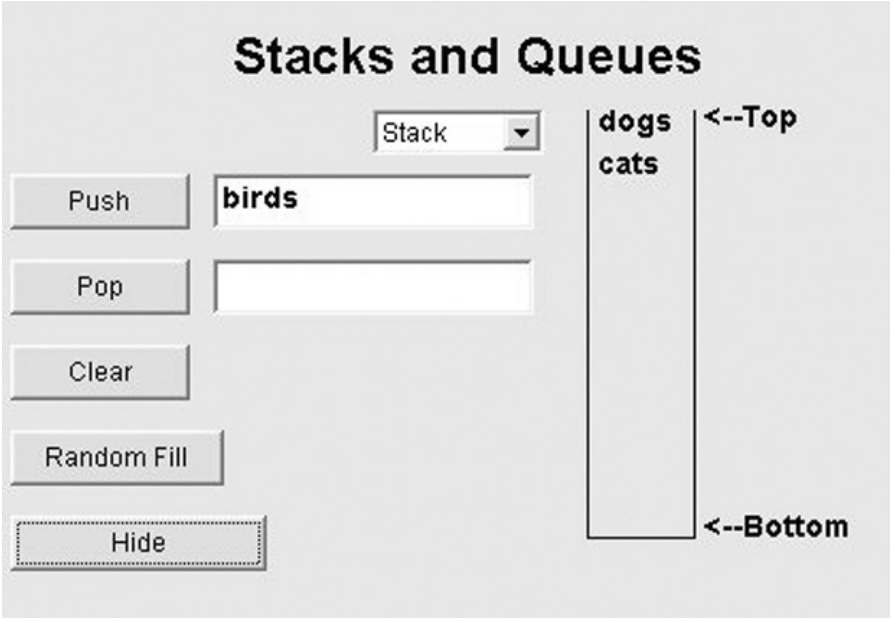
# Activity

## Part 1

Two commonly used abstract data types are stacks and queues, which are similar but are used in very different algorithms. Stacks are probably the most heavily employed workhorses of computing, after lists. This lab allows you to watch both stacks and queues at work.

Start the "Stackqueue" applet. This oddly named applet does not imply that there is a third abstract data type, called a "stackqueue." Rather, the applet combines stacks and queues for convenience' sake. At any given time, the data structure underneath the surface of the applet is either a stack or a queue. By the way, get used to spelling that word (queue.) It isn't really that hard, just remember Q -UE - UE. Two sets of UE and you've won the spelling bee!

The applet starts as a stack that doesn't show itself initially. Click on the *Show* button. Type "cats" into the textfield next to the *Push* button and either press the *Push* button or press the Enter key after typing "cats." This pushes the word onto the stack. Since it is initially empty, "cats" will be the only thing on the stack. Cats like company so push "dogs" onto the stack. Finally push "birds" on. Here's what the applet looks like just before you push "birds" into this happy menagerie:

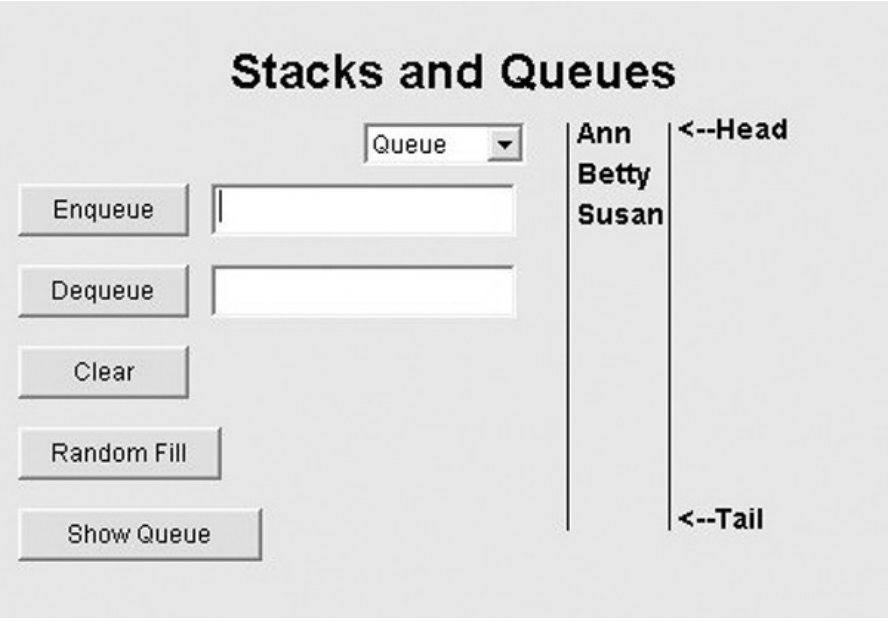



Remember that the first thing you push onto a stack is the last thing that pops out, like the spring-loaded plate well at a cafeteria mentioned on p. 308.

When you click the *Pop* button, the top thing is taken off the stack and appears in the textfield next to the *Pop* button. You can pop everything immediately by pressing *Clear*.

The applet can put a bunch of random numbers on the stack when you press the *Random Fill* button. Though numbers aren't as interesting as animals or names of friends (at least to most people), they are quick and easy to generate randomly.

Clear the stack, and then pull down the menu so that Queue appears. Notice how the picture changes so that there is a Head and Tail, instead of Top and Bottom. Also notice that the queue is open-ended, instead of closed at the bottom, like the stack. The buttons change, too, because Computer Scientists don't like to confuse us by saying that we push things onto queues and pop them off. Rather, they suggest that we enqueue and dequeue, or enter and delete, when using a queue.




Put three names onto the queue. In the picture above, we enqueued Ann, then Betty, then Susan, in that order. If this were a stack, Ann would be at the bottom and Susan would be at the top. But queues work differently.

When you insert something into a queue, it goes at the end of the line, just like you do when you check out at a grocery store. Whoever is at the head of the line will get checked out next . . . you just have to wait!

What is so different about stacks versus queues? Unfortunately, you will have to learn more about computers and algorithms before you see why both are necessary and not interchangeable. But here are a few hints.

Queues are used in simulations where you generate tasks that you want to finish later or send off to another processor. Operating systems use queues. They often reorder the elements in the queue according to a priority level so that the most important tasks get done first.

Stacks are indispensable to language processing. Researchers believe that we parse sentences by using stacks to remember what phrase modifies which word. Here's an example:

> I saw the dog belonging to the girl whom my sister who is studying at the school with the fabulous new library brought with her.

Admittedly, that is kind of awkward, but most English speakers would be able to understand it. They might argue about "whom" or "who," but let's not fight about grammar.

Here's how it works. There are successive levels of phrases and modifiers. Modifying phrases, either prepositional phrases like "at the school" or subordinate clauses like "who is studying . . ." follow the nouns they modify. As an English speaker listens to a sentence like the one above, she picks out the most important structures, and then fits the modifiers in, kind of like pickles and relish on a hamburger, nice but not essential. The attachment of phrases works backward like a stack.

Here's the way many linguists think we figure out the structure. The stack is constructed so that a phrase modifies the thing directly underneath it, in most cases.

| | | |
|---|---|---|
| top → | brought with *her*. | this is what my sister did |
| | with the fabulous new library | this phrase modifies school |
| | who is studying at the school | this phrase modifies sister |
| | whom my sister | this phrase modifies girl |
| | belonging to the girl | this phrase modifies dog |
| | the dog | this is what I saw |
| | I saw | this is the subject and main verb |

Notice how we disambiguate *her*, which could refer to any female, in particular my sister or the girl. Most English speakers would claim that *her* refers to my sister, not the girl. Linguists contend that this is evidence that we use a mental stack to process complex sentences.

Our applet has a *mystery* option, which is not some kind of mystical mixture of stacks and queues, but rather a random choice between the stack or the queue. If you are showing the stack or queue, click the *Hide* button. Otherwise, there's no more mystery. When you select mystery, the buttons change to *Add* and *Remove* so that they won't give away the identity of the mystery abstract data type.

Pull down the mystery option. The applet has chosen either a stack or a queue—we don't know which. Press the *Clear* button to empty it. Now Add "John," then add "Tony."

Here's the fun part. Click on *Remove*. What appears? Is it "John" or "Tony"? What does it mean if it is "John" instead of "Tony"? If you understand the difference between stacks and queues, the answer will be obvious. Just remember LIFO for stacks and FIFO for queues:

| | |
|---|---|
| *LIFO = Last In First Out* | *Stacks* |
| *FIFO = First In First Out* | *Queues* |

One last thing. If you try to remove something when the stack or queue is empty, what happens? Also, try to add many things in order to determine the maximum size of the stack or queue. Our applet has a fixed maximum size, but in real applications, programmers can change it.

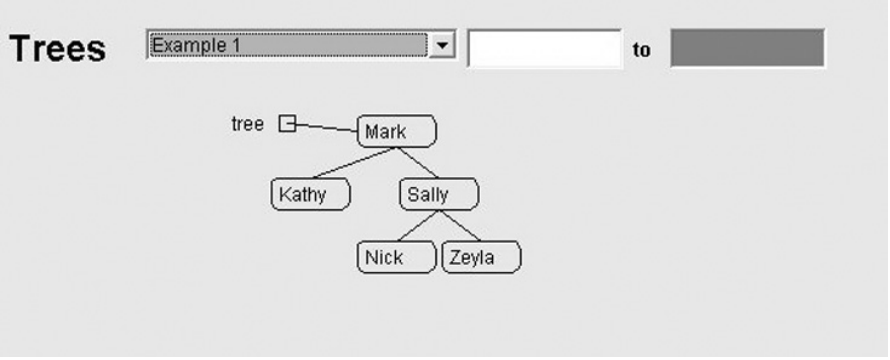
## Part 2

Trees are enormously important, both to provide shade and grow apples and cherries . . . Ooops, wrong trees! The use of the word "tree" to describe the abstract data type that looks more like a root system just shows how upside-down the view of the world as seen by Computer Scientists is. If they start to collaborate with botanists, watch out!

Trees are everywhere in Computer Science. Language processing uses trees, as do compilers of higher level languages like C and Java. You will find binary trees when you examine the playoff schedule for a sports event. Whichever of two teams or contestants who wins moves up the tree to the next level.

Your textbook discusses several types of trees and gives algorithms for examining and changing them. The "Trees" applet allows you to work with a tree and visualize it, but does not permit you to write computer programs to manipulate them. That will come along in later programming courses.

Start the "Trees" applet. Pull down the menu until "Example 1" appears. This creates a tree with five names, as shown below:

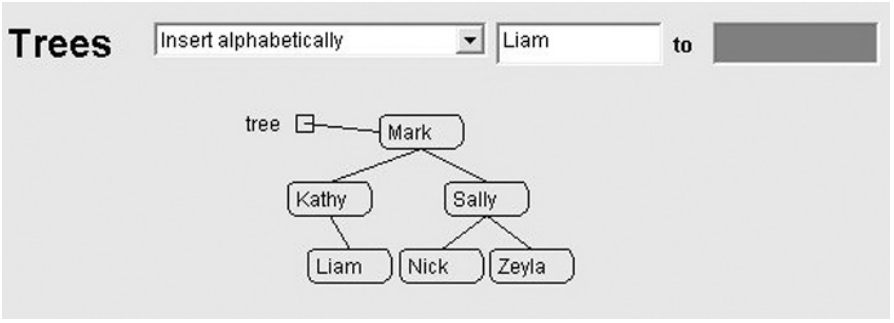



A typical computer program may have many trees, which is sometimes referred to as a "forest" (no kidding, and no logging!) In our applet, there is only one tree, referred to as simply "tree." Notice the little square next to it. That represents the pointer variable that contains the address of the top node, which is occupied by "Mark." Sadly, our applet doesn't have arrows, only lines, but the lines always point down, just like your textbook shows on pages 312–318.

The trees that the examples in this applet create are all binary trees, which means that each node has either no children, one child, or two children. Other trees permit many children. We can also talk about grandchildren nodes, though the term descendant is more common.

A special kind of binary tree is a binary search tree that arranges the nodes in such a way that they can be searched easily. Given a node, all of its left descendants have values that are less than it, numerically or alphabetically. "Kathy" is less than "Mark" because "Kathy" precedes "Mark" in the dictionary, so her node is in the left subtree of "Mark." Similarly, all the right descendants of a node have values greater than the node's data value.

Pull down the menu and select “Insert alphabetically.” Then type the name “Liam” into the textfield next to it and press return:



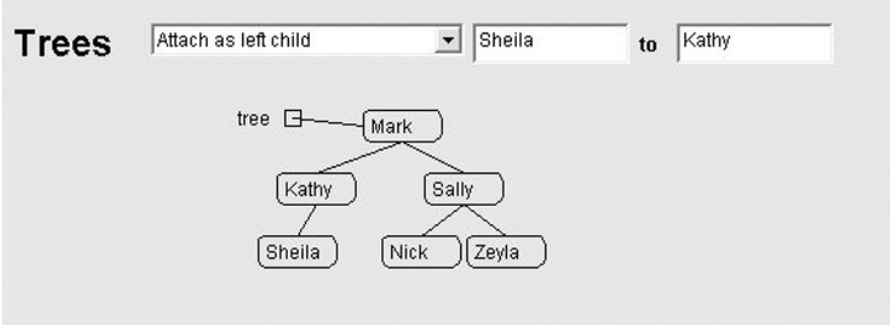

Notice that “Liam” comes between “Kathy” and “Mark” alphabetically. However, the “Mark” node already has two children so it is full. Thus, “Liam” has to go into the left subtree of “Mark.” But it can’t just replace “Kathy.” Instead, “Liam” has to be a child or grandchild, or great-grandchild of “Kathy.” Since “Kathy” has no other children at the moment, “Liam” is her right child because “L” comes after “K.”

Experiment with some more names to see where the applet places them. If you select Example 3, you will see a complete binary tree, one where every non-leaf node contains exactly two children. Again, this shows how fanciful and upside-down the Computer Science conception of a tree is . . . the leaves are at the bottom?

Select “Find” from the pull-down menu and type “Steve.” Watch how some of the nodes turn yellow as the search progresses. You can also type a name that is not in the tree to see how the applet responds.

One fabulous property of binary search trees is that they store a huge amount of information in such a way that you can get to the item you want quickly, unlike a list where you might have to look at everything in list. Notice how the search process descends quickly and doesn’t have to look at all the nodes in the tree.

You can attach any node to any existing node in this applet by using one of the two “Attach” options from the pull-down menu. However, this may screw up the alphabetical ordering by doing this. Let’s experiment. Select “Clear” and then “Example 1.” Now select “Attach as left child” and type “Sheila” in the first textfield and “Kathy” in the second. The second textfield is normally gray so that you can’t use it, only becoming usable when you select one of the two “Attach” options.




Our tree is still a binary tree, though not a complete one, but it is out of order. Can it find “Sheila” if you search for her? Try it. Why won’t it find her?

# Exercise 1

Name _____ Date _____

Section _____

1) Start the “Stackqueue” applet.

2) The purpose of this exercise is see how you could reverse a list of names. Push the three names “Abe,” “Betty,” and “Charles” onto the stack. When you pop the stack 3 times, what do you see in the textfield next to “Pop”?

3) Write an algorithm in pseudo-code or English that would describe how to use a stack to reverse any list. Write your answer below or type it up in a word processor.

4) A palindrome or RADAR word is one that has the same sequence of letters if you read it from right to left or left to right. Describe how you could use a stack in conjunction with the original list to determine if a sequence is a palindrome. Write your answer below or type it up in a word processor. (Hint: Your letters exist in a list. Use one stack and make copies of the letters. Then pop the stack and compare.)

# Exercise 2

Name _____ Date _____

Section _____

1) Start the “Stackqueue” applet.
2) The purpose of this exercise is to figure out whether the mystery data structure is a stack or a queue. Click on *Clear*, then *Random Fill*. Select “mystery” from the pull-down menu.
3) Click once on *Remove*. Whatever number appears next to the *Remove* button is what you now type into the textfield next to *Add*. Then click on *Add*.
4) Now click on *Remove*. What value appears? Is it the same one or different? What does this mean? Is your mystery object a stack or a queue? Take a screenshot and write your answer on the paper. You can then *Show* to confirm your conclusion.

# Exercise 3

Name _____ Date _____

Section _____

1) Start the "Trees" applet.
2) The purpose of this exercise is to build a binary search tree. Select "Insert alphabetically" from the pull-down menu. Then type the following flowers into the textfield, pressing return after each one.

   Lily

   Rose

   Daffodil

   Tulip

   Petunia
3) Take a screenshot. Now perform a manual search for "Poppy." Put a check mark next to each node that you visit during your search. You can confirm your ideas by selecting "Find" from the pull-down and asking the applet to search for "Poppy."
4) If you were to manually insert "Poppy" into this tree, what would you put into the boxes and which option would you select?

   ____ "Attach as left child" _____ to _____

   ____ "Attach as right child"

   Check one blank on the left of "Attach…" and fill in the two blanks surrounding "to."

# Exercise 4

Name _____ Date _____

Section _____

1) Start the "Trees" applet.
2) The purpose of this exercise is to build a binary search tree but choosing different orders for inserting the nodes.
3) Type in the following flower names in the given order. (If you don't want to type a lot, just use the first letter of each name.) Take a screenshot when done.

   Aster

   Bluebell

   Coreopsis

   Daisy

   Echinacea

   Fern

   Gladiolus

4) What does your tree look like? Describe it in words below.

5) What generalization can you make about inserting elements from a sorted list into a tree?

6) Could you convince someone that a list is a special kind of tree?

7) Clear the tree and insert the flowers in some other order so that the tree is *balanced* and *complete*. Remember that this means that all nodes except the leaf nodes have exactly two children. After you are successful, take a screenshot. Also write down the exact order that you used to get it to look this way.

8) Is there more than one order that you could have used that would have created the same balanced, complete tree? Why are multiple different orders possible?

# Exercise 5

Name _____ Date _____

Section _____

1) Start the “Trees” applet.
2) The purpose of this exercise is to investigate what happens when you have duplicate nodes. Choose “Example 2.”
3) Select “Insert Alphabetically.” Then type “Mark” in the textfield and press Return. Take a screenshot.

4) Insert another name that is already in the tree. What does this applet do when you ask it to enter a name that is already there?

5) Are there alternative strategies that the applet could have used? List two more.
   a) _____
   b) _____
6) Does “Find” still work? Try to find a name, like “Mark,” that is represented by two nodes. What happens? Describe.

## Deliverables

Turn in your hand-written sheets showing your answers to the experiments.

## Deeper Investigation

Just as in sorting, there are many search algorithms besides sequential and binary searching. For example, there is the tree searching algorithm presented on p. 315 of your textbook). Think about how you search for a name in a telephone book. You don't need to live in New York City or Los Angeles to realize that *nobody* in their right mind uses sequential search on the telephone book! But just exactly what do people do? What algorithm do they employ? Is it binary search? Can you describe the algorithm in English, even if it is too hard to write out in code?

Think about searching the telephone book again. What kinds of additional information are present to help humans find names in the phone book? How are phone books organized? (Hint: Look at the tops of pages in the phone book.)

Finally, think about finding information in your school's library. You probably have a computer system with an online catalog, but not too long ago, libraries had huge filing cabinets full of $3 \times 5$ index cards. (In fact, some folks get downright nostalgic for those drawers of musty-smelling cards....) Such indexes now live on (relatively scent-free) computer hard disks.

Think about the steps you would have to go through to find a specific piece of information or a quotation in your library. What indexes would you search? What happens when the index leads you to the book itself? What then? How might all this change in the upcoming years as computerization advances?