6.4 Exercises

Reinforcement

R-6.1 Suppose an initially empty stack S has performed a total of 25 push operations, 12 top operations, and 10 pop operations, 3 of which returned null to indicate an empty stack. What is the current size of S?

R-6.2 Had the stack of the previous problem been an instance of the ArrayStack class, from Code Fragment 6.2, what would be the final value of the instance variable t?

R-6.3 What values are returned during the following series of stack operations, if executed upon an initially empty stack? push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().

R-6.4 Implement a method with signature transfer(S, T) that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T.

R-6.5 Give a recursive method for removing all the elements from a stack.

R-6.6 Give a precise and complete definition of the concept of matching for grouping symbols in an arithmetic expression. Your definition may be recursive.

R-6.7 Suppose an initially empty queue Q has performed a total of 32 enqueue operations, 10 first operations, and 15 dequeue operations, 5 of which returned null to indicate an empty queue. What is the current size of Q?

R-6.8 Had the queue of the previous problem been an instance of the ArrayQueue class, from Code Fragment 6.10, with capacity 30 never exceeded, what would be the final value of the instance variable f?

R-6.9 What values are returned during the following sequence of queue operations, if executed on an initially empty queue? enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue().

R-6.10 Give a simple adapter that implements the stack ADT while using an instance of a deque for storage.

R-6.11 Give a simple adapter that implements the queue ADT while using an instance of a deque for storage.

R-6.12 What values are returned during the following sequence of deque ADT operations, on an initially empty deque? addFirst(3), addLast(8), addLast(9), addFirst(1), last(), isEmpty(), addFirst(2), removeLast(), addLast(7), first(), last(), addLast(4), size(), removeFirst(), removeFirst().

R-6.13 Suppose you have a deque D containing the numbers $(1,2,3,4,5,6,7,8)$, in this order. Suppose further that you have an initially empty queue Q. Give a code fragment that uses only D and Q (and no other variables) and results in D storing the elements in the order $(1,2,3,5,4,6,7,8)$.

R-6.14 Repeat the previous problem using the deque D and an initially empty stack S.

R-6.15 Augment the ArrayQueue implementation with a new rotate( ) method having semantics identical to the combination, enqueue(dequeue( )). But, your implementation should be more efficient than making two separate calls (for example, because there is no need to modify the size).

Creativity

C-6.16 Suppose Alice has picked three distinct integers and placed them into a stack S in random order. Write a short, straightline piece of pseudocode (with no loops or recursion) that uses only one comparison and only one variable x, yet that results in variable x storing the largest of Alice's three integers with probability $2/3$. Argue why your method is correct.

C-6.17 Show how to use the transfer method, described in Exercise R-6.4, and two temporary stacks, to replace the contents of a given stack S with those same elements, but in reversed order.

C-6.18 In Code Fragment 6.8 we assume that opening tags in HTML have form `<name>`, as with `<li>`. More generally, HTML allows optional attributes to be expressed as part of an opening tag. The general form used for expressing an attribute is `<name attribute1="value1" attribute2="value2">`; for example, a table can be given a border and additional padding by using an opening tag of `<table border="3" cellpadding="5">`. Modify Code Fragment 6.8 so that it can properly match tags, even when an opening tag may include one or more such attributes.

C-6.19 ***Postfix notation*** is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if "(exp_1) **op** (exp_2)" is a normal fully parenthesized expression whose operation is **op**, the postfix version of this is "$pexp_1$ $pexp_2$ **op**", where $pexp_1$ is the postfix version of exp_1 and $pexp_2$ is the postfix version of exp_2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of "$((5+2)*(8-3))/4$" is "5 2 + 8 3 − ∗ 4 /". Describe a nonrecursive way of evaluating an expression in postfix notation.

C-6.20 Suppose you have three nonempty stacks R, S, and T. Describe a sequence of operations that results in S storing all elements originally in T below all of S's original elements, with both sets of those elements in their original order. The final configuration for R should be the same as its original configuration. For example, if $R = (1,2,3)$, $S = (4,5)$, and $T = (6,7,8,9)$, when ordered from bottom to top, then the final configuration should have $R = (1,2,3)$ and $S = (6,7,8,9,4,5)$.

C-6.21 Describe a nonrecursive algorithm for enumerating all permutations of the numbers $\{1,2,\ldots,n\}$ using an explicit stack.

C-6.22 Alice has three array-based stacks, A, B, and C, such that A has capacity 100, B has capacity 5, and C has capacity 3. Initially, A is full, and B and C are empty. Unfortunately, the person who programmed the class for these stacks made the push and pop methods private. The only method Alice can use is a static method, dump(S,T), which transfers (by iteratively applying the private pop and push methods) elements from stack S to stack T until either S becomes empty or T becomes full. So, for example, starting from our initial configuration and performing dump(A,C) results in A now holding 97 elements and C holding 3. Describe a sequence of dump operations that starts from the initial configuration and results in B holding 4 elements at the end.

C-6.23 Show how to use a stack S and a queue Q to generate all possible subsets of an n-element set T nonrecursively.

C-6.24 Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may only use S, Q, and a constant number of other primitive variables.

C-6.25 Describe how to implement the stack ADT using a single queue as an instance variable, and only constant additional local memory within the method bodies. What is the running time of the push(), pop(), and top() methods for your design?

C-6.26 When implementing the ArrayQueue class, we initialized $f = 0$ (at line 5 of Code Fragment 6.10). What would happen had we initialized that field to some other positive value? What if we had initialized it to -1?

C-6.27 Implement the clone( ) method for the ArrayStack class. (See Section 3.6 for a discussion of cloning data structures.)

C-6.28 Implement the clone( ) method for the ArrayQueue class. (See Section 3.6 for a discussion of cloning data structures.)

C-6.29 Implement a method with signature concatenate(LinkedQueue<E> Q2) for the LinkedQueue<E> class that takes all elements of Q2 and appends them to the end of the original queue. The operation should run in $O(1)$ time and should result in Q2 being an empty queue.

C-6.30 Give a pseudocode description for an array-based implementation of the double-ended queue ADT. What is the running time for each operation?

C-6.31 Describe how to implement the deque ADT using two stacks as the only instance variables. What are the running times of the methods?

C-6.32 Suppose you have two nonempty stacks S and T and a deque D. Describe how to use D so that S stores all the elements of T below all of its original elements, with both sets of elements still in their original order.

C-6.33 Alice has two circular queues, C and D, which can store integers. Bob gives Alice 50 odd integers and 50 even integers and insists that she stores all 100 integers in C and D. They then play a game where Bob picks C or D at random and then applies the `rotate()` method to the chosen queue a random number of times. If the last number to be rotated at the end of this game is odd, Bob wins. Otherwise, Alice wins. How can Alice allocate integers to queues to optimize her chances of winning? What is her chance of winning?

C-6.34 Suppose Bob has four cows that he wants to take across a bridge, but only one yoke, which can hold up to two cows, side by side, tied to the yoke. The yoke is too heavy for him to carry across the bridge, but he can tie (and untie) cows to it in no time at all. Of his four cows, Mazie can cross the bridge in 2 minutes, Daisy can cross it in 4 minutes, Crazy can cross it in 10 minutes, and Lazy can cross it in 20 minutes. Of course, when two cows are tied to the yoke, they must go at the speed of the slower cow. Describe how Bob can get all his cows across the bridge in 34 minutes.

Projects

P-6.35 Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value.

P-6.36 When a share of common stock of some company is sold, the ***capital gain*** (or, sometimes, loss) is the difference between the share's selling price and the price originally paid to buy it. This rule is easy to understand for a single share, but if we sell multiple shares of stock bought over a long period of time, then we must identify the shares actually being sold. A standard accounting principle for identifying which shares of a stock were sold in such a case is to use a FIFO protocol—the shares sold are the ones that have been held the longest (indeed, this is the default method built into several personal finance software packages). For example, suppose we buy 100 shares at $20 each on day 1, 20 shares at $24 on day 2, 200 shares at $36 on day 3, and then sell 150 shares on day 4 at $30 each. Then applying the FIFO protocol means that of the 150 shares sold, 100 were bought on day 1, 20 were bought on day 2, and 30 were bought on day 3. The capital gain in this case would therefore be $100 \cdot 10 + 20 \cdot 6 + 30 \cdot (-6)$, or $940. Write a program that takes as input a sequence of transactions of the form "`buy` x `share(s) at $`y `each`" or "`sell` x `share(s) at $`y `each`," assuming that the transactions occur on consecutive days and the values x and y are integers. Given this input sequence, the output should be the total capital gain (or loss) for the entire sequence, using the FIFO protocol to identify shares.

P-6.37 Design an ADT for a two-color, double-stack ADT that consists of two stacks—one "red" and one "blue"—and has as its operations color-coded versions of the regular stack ADT operations. For example, this ADT should support both a `redPush` operation and a `bluePush` operation. Give an efficient implementation of this ADT using a single array whose capacity is set at some value N that is assumed to always be larger than the sizes of the red and blue stacks combined.

P-6.38 The introduction of Section 6.1 notes that stacks are often used to provide "undo" support in applications like a Web browser or text editor. While support for undo can be implemented with an unbounded stack, many applications provide only ***limited*** support for such an undo history, with a fixed-capacity stack. When push is invoked with the stack at full capacity, rather than throwing an exception, a more typical semantic is to accept the pushed element at the top while "leaking" the oldest element from the bottom of the stack to make room. Give an implementation of such a LeakyStack abstraction, using a circular array.

P-6.39 Repeat the previous problem using a singly linked list for storage, and a maximum capacity specified as a parameter to the constructor.

P-6.40 Give a complete implementation of the Deque ADT using a fixed-capacity array, so that each of the update methods runs in $O(1)$ time.

Chapter Notes

We were introduced to the approach of defining data structures first in terms of their ADTs and then in terms of concrete implementations by the classic books by Aho, Hopcroft, and Ullman [5, 6]. Exercises C-6.22, C-6.33, and C-6.34 are similar to interview questions said to be from a well-known software company. For further study of abstract data types, see Liskov and Guttag [67] and Demurjian [28].