4.27. Programming Exercises

1. Modify the infix-to-postfix algorithm so that it can handle errors.

2. Modify the postfix evaluation algorithm so that it can handle errors.

3. Implement a direct infix evaluator that combines the functionality of infix-to-postfix conversion and the postfix evaluation algorithm. Your evaluator should process infix tokens from left to right and use two stacks, one for operators and one for operands, to perform the evaluation.

4. Turn your direct infix evaluator from the previous problem into a calculator.

5. Implement the `Queue` ADT, using a list such that the rear of the queue is at the end of the list.

6. Design and implement an experiment to do benchmark comparisons of the two queue implementations. What can you learn from such an experiment?

7. It is possible to implement a queue such that both enqueue and dequeue have $O(1)$ performance *on average*. In this case it means that most of the time enqueue and dequeue will be $O(1)$ except in one particular circumstance where dequeue will be $O(n)$.

8. Consider a real life situation. Formulate a question and then design a simulation that can help to answer it. Possible situations include:
   - Cars lined up at a car wash
   - Customers at a grocery store check-out
   - Airplanes taking off and landing on a runway
   - A bank teller

   Be sure to state any assumptions that you make and provide any probabilistic data that must be considered as part of the scenario.

9. Modify the Hot Potato simulation to allow for a randomly chosen counting value so that each pass is not predictable from the previous one.

(DiscussionQuestions.html) (../Recursion/toctr

10. Implement a radix sorting machine. A radix sort for base 10 integers is a mechanical sorting technique that utilizes a collection of bins, one main bin and 10 digit bins. Each bin acts like a queue and maintains its values in the order that they arrive. The algorithm begins by placing each number in the main bin. Then it considers each value digit by digit. The first value is removed and placed in a digit bin corresponding to the digit being considered. For example, if the ones digit is being considered, 534 is placed in digit bin 4 and 667 is placed in digit bin 7. Once all the values are placed in the corresponding digit bins, the values are collected from bin 0 to bin 9 and placed back in the main bin. The process continues with the tens digit, the hundreds, and so on. After the last digit is processed, the main bin contains the values in order.

11. Another example of the parentheses matching problem comes from hypertext markup language (HTML). In HTML, tags exist in both opening and closing forms and must be balanced to properly describe a web document. This very simple HTML document:

```
<html>
   <head>
      <title>
         Example
      </title>
   </head>

   <body>
      <h1>Hello, world</h1>
   </body>
</html>
```

is intended only to show the matching and nesting structure for tags in the language. Write a program that can check an HTML document for proper opening and closing tags.

12. Extend the program from Listing 2.15 to handle palindromes with spaces. For example, I PREFER PI is a palindrome that reads the same forward and backward if you ignore the blank characters.

13. To implement the `length` method, we counted the number of nodes in the list. An alternative strategy would be to store the number of nodes in the list as an additional piece of data in the head of the list. Modify the `UnorderedList` class to include this information and rewrite the `length` method.

14. Implement the `remove` method so that it works correctly in the case where the item is not in the list.

(DiscussionQuestions.html)

(../Recursion/toctr

15. Modify the list classes to allow duplicates. Which methods will be impacted by this change?

16. Implement the __str__ method in the UnorderedList class. What would be a good string representation for a list?

17. Implement __str__ method so that lists are displayed the Python way (with square brackets).

18. Implement the remaining operations defined in the UnorderedList ADT (append, index, pop, insert).

19. Implement a slice method for the `UnorderedList` class. It should take two parameters, `start` and `stop`, and return a copy of the list starting at the `start` position and going up to but not including the `stop` position.

20. Implement the remaining operations defined in the OrderedList ADT.

21. Consider the relationship between Unordered and Ordered lists. Is it possible that inheritance could be used to build a more efficient implementation? Implement this inheritance hierarchy.

22. Implement a stack using linked lists.

23. Implement a queue using linked lists.

24. Implement a deque using linked lists.

25. Design and implement an experiment that will compare the performance of a Python list with a list implemented as a linked list.

26. Design and implement an experiment that will compare the performance of the Python list based stack and queue with the linked list implementation.

(DiscussionQuestions.html) (../Recursion/toctr

27. The linked list implementation given above is called a singly linked list because each node has a single reference to the next node in sequence. An alternative implementation is known as a doubly linked list. In this implementation, each node has a reference to the next node (commonly called

next) as well as a reference to the preceding node (commonly called back). The head reference also contains two references, one to the first node in the linked list and one to the last. Code this implementation in Python.

28. Create an implementation of a queue that would have an average performance of O(1) for enqueue and dequeue operations.

user not logged in

(DiscussionQuestions.html)

(../Recursion/toctr