7.3 Practice Programs

Sample solutions to these problems will be included by the beginning of 2013 at the following website:

http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3223/book/

1) Write a program that reads in a regular text file as input and outputs how many of each letter appeared in that file.

2) Write a program that reads in a text file of test scores and prints out a histogram of the scores. The file format is as follows: the first line of the file contains a single positive integer, *n*, representing the number of test scores. The following *n* lines will contain one test score each. Each of these test scores will be a non-negative integer less than or equal to 100. The histogram should have one row on it for each distinct test score in the file, followed by one star for each test score of that value. For example, if the tests scores in the file were 55, 80, 80, 95, 95, 95 and 98, the output to the screen should look like:

```
55*
80**
95***
98*
```

3) Write a program that prints out a random permutation of the numbers 1 through *n*, where *n* is a positive integer entered by the user.

4) Write a program that reads a set of dart throws from a user and computes the user's score. Assume that the user enters 21 dart throws and each throw is a single integer in between 1 and 20, inclusive. To compute the user's score, look at each number in between 15 and 20, inclusive that the user threw more than three times, and add up the points of the throws, after the third throw. For example, if the user throws 5 20s, 3 19s, 4 18s, 6 14s and 2 1s, then the user's score is 2x20+1x18 = 58, since the user threw 2 more 20s than 3 and 1 more 18 than 3. The 14s don't count since only 15 through 20 count for points. Extend this program by reading in the throw data from a file. Make up your own file format.

5) Write a program that reads in 10 distinct integers from the user into an array in the order entered by the user and computes the number of inversions in the array. An inversion in an array is when a bigger value comes before a smaller value. The list 3, 6, 2, 1, 5, 4 has the following 8 inversions: (3, 2), (3, 1), (6, 2), (6, 1), (6, 5), (6, 4), (2, 1), and (5, 4).

6) A permutation is an ordering of a given set of numbers. For example, the 6 permutations of 0, 1 and 2 are (0,1,2), (0,2,1), (1,0,2), (1,2,0), (2,0,1) and (2,1,0). Write a program that asks the user for a positive integer n and prints out a random permutation of the integers from 0 to n-1.

7) One can view a permutation of the integers 0, 1, 2, ..., n-1 as a function, For example, for n = 3, the permutation 2, 0, 1 can be viewed as a function f where f(0) = 2, f(1) = 0 and f(2) = 1. Consider calculating f(f(x)), for this function x:

f(f(0)) = f(2) = 1 f(f(1)) = f(0) = 2 f(f(2)) = f(1) = 0

Now, we'll calculate f(f(f(x))):

f(f(f(0))) = f(1) = 0 f(f(f(1))) = f(2) = 1 f(f(f(2))) = f(0) = 2

Notice that this function is the identity function. Namely, f(x) = x, for each input integer x.

For all given permutations, if we apply it enough times in succession as a function (function composition), we will return to the identity function, or the permutation of the numbers in order.

Write a program that asks the user to enter a positive integer n, and then ask them to enter a permutation of the values 0, 1, 2, ..., n-1. Then, calculate the number of times this permutation needs to be applied to itself until it produces the identity function.

8) A sultan has n precious stones that he wants to place in n boxes. The boxes are labeled 0 through *n-1* and arranged in a circle, in clockwise order. (Thus, the box that follows box number *n-1* is box number 0.) He will place the first stone, stone number 1, in box 0. From there, he'll jump *k* boxes ahead and place stone number 2 in that box. For example, if *k* is 3, the second stone would go in box 3. The sultan will either be able to place each stone in a different box, OR, he'll eventually attempt to put a stone into a box that already has one. In the latter case, the sultan will immediately stop placing the stones in boxes. After this process is done, print out a list of each non-empty box in order, stating which numbered stone it has. For example, if n = 5 and k = 2, the output should be as follows:

```
Box 0 contains stone 1.
Box 1 contains stone 4.
Box 2 contains stone 2.
Box 3 contains stone 5.
Box 4 contains stone 3.
```

Alternatively, if n = 6 and k = 4, the output should be:

```
Box 0 contains stone 1.
Box 2 contains stone 3.
Box 4 contains stone 2.
```

Bonus question: under what conditions does the sultan succeed in placing all of the stones in the boxes?

9) The sultan has decided that he wants a new system that will always succeed in placing the stones in the boxes. Thus, when he skips ahead *k* boxes, he'll ONLY count empty boxes. Thus, in the first example given in problem 8, after placing stone 3 in box 4, he will skip over the empty box 2 and place stone 4 in box 4. Finally, he'll skip over the empty box 2 only to arrive at it again to place stone 5. Using this new system, the output for the case where n = 6 and k = 4 is:

```
Box 0 contains stone 1.
Box 1 contains stone 6.
Box 2 contains stone 5.
Box 3 contains stone 3.
Box 4 contains stone 2.
Box 5 contains stone 4.
```

10) Write a program that reads in information about a set of pizza orders from "pizza.txt" and calculates the total price of each order. In particular, the input file ("pizza.txt") will be formatted as follows:

1. The first line of the input file will contain a single positive integer, *n* (*n* < *100)*, specifying the number of different pizza orders.

2. The next *n* lines will each contain one positive real number specifying the price for that corresponding pizza. (The first line will have the price for pizza 0, the next line for pizza 1, etc.)

3. The following line will contain a single positive integer *k* (*k* < 50) representing the number of orders you have to evaluate.

4. The next *k* lines will contain information about each of the *k* orders with one order per line.

5. Each of these lines will contain *n* non-negative integers, representing how many of those pizzas, respectively are in the order. (For example, if n=5 and the line contains the values 0 0 6 0 9, then the order contains 6 slices of pizza #2 and 9 slices of pizza #4.)

For each of the *k* test cases, output a line with the following format:

```
On day X, you will spend $YY.YY on pizza.
```

where `X` is the test case number (starting with 1), and YY.YY is the price of the pizza displayed with exactly 2 digits after the decimal. (Note: The price may exceed 100 dollars, so YY simply represents a dollar amount and not the exact number of digits in that dollar amount.)

Sample Input File (pizza.txt)

```
5
3.00
3.50
4.50
5.00
6.00
3
1 1 1 1 1
0 0 2 1 6
1 0 3 2 3
```

Sample Output (Corresponding to Sample Input File)

```
On day 1, you will spend $22.00 on pizza.
On day 2, you will spend $50.00 on pizza.
On day 3, you will spend $44.50 on pizza.
```