

Control Flow Exercises

See the submission instructions in the handout with instructions for exercises that I distributed January 26th. Here I'll just remind you that this is not homework and that I will be happy to give you hints if you get stuck on any of the problems.

I recommend problems 3, 5, 9, 10, and the first paragraph of problem 14. There aren't that many difficult problems on this handout. The only difficulty is that some of the problems are long. I think problem 8 is neat. One complication is in the "more ambitious" version of problem 11, and I'm curious to see your solutions to problem 13. Problems 1 and 4 give some more insight into the mechanics of Java. Problem 6 should remind you of something on the previous handout.

1 Decision Exercises

1) Here are two code fragments. Put them in two separate programs and run those two programs. Either program should let the user enter a line of text, store that line in a variable called `line`, and then run one of the two fragments. Do those two programs behave the same on every input? Why or why not?

Fragment 1:

```
if (!line.isEmpty() && line.charAt(0) == 'c') {
    System.out.println("Line starts with c");
} else {
    System.out.println("Line doesn't start with c");
}
```

Fragment 2:

```
if (line.charAt(0) == 'c' && !line.isEmpty()) {
    System.out.println("Line starts with c");
} else {
    System.out.println("Line doesn't start with c");
}
```

2) A chess board has eight rows labeled 1 through 8 and eight columns labeled *a* through *h*. This allows us to uniquely identify each square using the corresponding row number and column letter.

A knight makes L-shaped moves on the chess board. It moves two squares in one direction (up, down, left, or right), and then one square in a direction perpendicular to the starting direction. For example, a knight that's on e2 can move to c1, c3, d4, f4, g3, and g1.

Write a program that takes as input the position of the knight on the chess board and the coordinates of the intended destination. Your program shall print whether a knight can move from its current position to the intended destination. You can also give the user a list of squares where the knight can move.

Note: You may find it more convenient to use a pair of integers to store the knight's position and intended destination, and prompt the user to enter integers only. You can modify your program later to take inputs as specified in the previous paragraph.

3) Write a code fragment that checks whether the string `str` starts with an uppercase letter. There are at least two easy ways to do this that I can see off the top of my head. How many can you come up with?

4) What is the output from the following line?

```
System.out.println((int)'a');
```

Based on your findings, what do you think the values of the following boolean expressions are?

a) `'Z' > 'a'`

b) `'z' > 'A'`

Attempt to figure this out without writing a program that checks the truth of those two expressions. You are welcome to write other programs to help you answer the question.

2 Repetition Exercises

5) Write a program that prints the lyrics of the song below. Ignore the rules of English grammar on your first try and test your program. After you verify that the program works correctly, modify your code to make the output grammatically correct. Also, be prepared to sing the song in lecture.

```
10 bottles of beer on the wall
10 bottles of beer
take a beer down, pass it around
9 bottles of beer on the wall
...
2 bottles of beer on the wall
2 bottles of beer
take a beer down, pass it around
1 bottle of beer on the wall
1 bottle of beer on the wall
1 bottle of beer
take a beer down, pass it around
no bottles of beer on the wall
```

6) Will the following loop terminate? Explain.

```
for (int i = 10; i > 0; i++)
{
    System.out.println("Hello");
}
```

7) Write a program that prints a string backwards.

8) *This question looks very similar to question 7, but requires slightly different tools.*

You have an integer variable `x`. Write a program that produces another integer, `y`, whose digits are like those in `x`,

except in reversed order. For example, if x is 123454, the value of y should be 454321.

9) Write a program that takes two strings as input, and combines them into one string where the characters are interleaved (like when you close a zipper). If you run out of characters in one of the strings, just put the remainder of the other string at the end. For example, if the two strings entered by the user are "12345" and "abcdefg", the resulting string should be "1a2b3c4d5efg".

10) Write a program that takes a string as input, and prints the characters from the middle to the outside. That is, first print the middle character (or the left one and then the right one if the string has even length), then print the characters one to the left of the middle and one to the right of the middle, and so on. The last two characters that get printed are the first and the last character, in that order. For example, if the input is "12345", your program should print "32415", and if the input is "123456", your program should print "342516".

11) Write a program that reads positive integers from the user until the user enters zero (for the more adventurous souls, until the user enters the string "zero"). When the user is done entering numbers, print out the average, the sum, the largest number, and the smallest number the user entered. Don't count the *sentinel value* zero. Validate that the user is entering non-negative integers, and yell at the user if he or she enters something invalid.

12) Write a program that takes one line as input, with words separated by one or more spaces (for example: "hello computer science ten bottles of beer on the wall"), and prints the shortest word. For the string above, the program could print "on".

13) Write a program that finds the longest substring two strings have in common and prints it out. For example, the longest common substring in the strings "computer" and "dumpster" is "ter".

14) Write code that prints various shapes from characters. In class we will write code that prints a triangle. Print that same triangle, but rotated 90 degrees, then 180 degrees, and then 270 degrees. The outputs for triangles of size 3 are shown in Figure 1.

```
x      x  xxx  xxx
xx     xx  xx  xx
xxx    xxx  x  x
```

Figure 1: Your output (from left to right): no rotation, rotation by 90 degrees, 180 degrees, and 270 degrees.

If you want to do some more *ASCII art*, print a square from 'x' characters, with '+' characters along the diagonal from the upper left corner to the lower right corner, or with different characters above and below the diagonal.

15) It is possible to approximate the square root of any positive n using the sequence x_0, x_1, x_2, \dots with $x_0 = n$ and $x_{k+1} = \frac{1}{2} \cdot (x_k + \frac{n}{x_k})$. Implement this approximation using a loop. The user will enter n and the value of x_0 .

You will never get \sqrt{n} exactly, so you should terminate the loop when the difference between the last two terms x_i and x_{i+1} is some small number, say $\epsilon = 0.00001$. How does the number of steps vary if you change the value of ϵ ?

If you feel mathematically strong and know the right tools, you can prove that the limit of the sequence x_0, x_1, \dots is indeed \sqrt{n} . If you feel even more ambitious, modify the algorithm to get any root of n requested by the user.

16) *The gambler's ruin problem is a very interesting problem in statistics, but appears in other areas as well. You can read about it on Wikipedia.*

We have two players. Player 1 has n_1 pennies, and player 2 has n_2 pennies. In each round, a coin is flipped. We get heads and tails with equal probability (although we can generalize it to having heads with probability p and tails with probability $1 - p$). If the flip ends up heads, player 1 gives a penny to player 2; otherwise, player 2 gives a penny to player 1. For various values of n_1 and n_2 , find the average number of coin flips it takes before one player goes broke. In the cases when $n_1 \neq n_2$, how often does the player who started with more pennies go broke?

17) *This problem is a modification of a programming project at the university where I got my bachelor's degree. It shows you that even though you've had only a few weeks of intro programming, you can already start doing interesting things and that you are not restricted to toy examples designed to give you programming practice.*

Let's model predator-prey relationships and corresponding changes in population sizes. The population sizes may repeat after some number of years, which gives rise to the study of *population cycles*. Suppose we have some number of rabbits and some number of foxes in a habitat. We would like to know how these populations change as time goes by.

In one simple model, rabbits have a reproduction rate of R_r , foxes have a death rate D_f , and there's also an interaction constant I_r for rabbits and I_f for foxes. We use these to calculate the new population sizes from the previous ones using the following formulas, where R_{old} and F_{old} are the old population sizes of rabbits and foxes, respectively, and R_{new} and F_{new} are the new population sizes.

$$\begin{aligned} R_{new} &= R_{old} + B_r \cdot R_{old} - I_r \cdot F_{old} \\ F_{new} &= F_{old} - D_f \cdot F_{old} + I_f \cdot R_{old} \end{aligned}$$

If a population ever gets below 1, the species goes extinct.

Write a program that simulates this interaction. Find some values of the starting populations and interaction constants where the population sizes repeat periodically, and some where a species goes extinct.