Homework 0: Recursion

cs230
Spring 2002

Allen B. Downey
Computer Science Department

Due: February 4

The purpose of this assignment is to review recursion and to write some programs that work with Strings. Please read Chapters 5 and 7 of the text, and the documentation of the String class.

All methods on this assignment should be class methods. We'll get to object methods in a few weeks.

A little number theory

1. Create a directory named `hw00` and `cd` into it. Download a copy of `Hello.java` from

   `http://rocky.wellesley.edu/cs230/code/hw00/Hello.java`

   and make sure you can compile and run it.

2. Make a copy of `Hello.java` named `Recurse.java`. Edit it with the text editor of your choice and change the name of the class to `Recurse`. Save the modified file and make sure you can compile and run it.

3. Add a class method called `power` to `Recurse.java`. It should compute x^n using the following recursive definition of exponentiation:

$$\begin{aligned} x^0 &= 1.0 \quad \text{for all } x \\ x^n &= x * x^{n-1} \quad \text{for } n > 0 \end{aligned} \tag{1}$$

   In the comments of the method, indicate how many recursions it requires as a function of x and n.

4. In `main`, write a few lines of code that invoke `power`. Compile and run the program and make sure `power` works correctly.

5. Write an improved version of `power` called `morePower` that takes advantage of the following recurrence:

$$\begin{aligned} x^n &= x^{n/2} \cdot x^{n/2} \quad \text{for even values of } n \\ x^n &= x \cdot x^{n/2} \cdot x^{n/2} \quad \text{for odd values of } n \end{aligned} \tag{2}$$

   In the comments of the method, indicate how many recursions it requires as a function of x and n.

6. Euclid's Algorithm is a well-known algorithm for computing the greatest common divisor (GCD) of two numbers. It appears in Euclid's *Elements* (Book 7, ca. 300 B.C.). It may be the oldest nontrivial algorithm.

   The algorithm is based on the observation that, if r is the remainder when a is divided by b, then the common divisors of a and b are the same as the common divisors of b and r. Thus we can use the equation

   ```
   GCD(a, b) = GCD(b, r)
   ```

   to successively reduce the problem of computing a GCD to the problem of computing the GCD of smaller and smaller pairs of integers. For example,

   ```
   GCD(36, 20) = GCD(20, 16) = GCD(16, 4) = GCD(4, 0) = 4
   ```

   implies that the GCD of 36 and 20 is 4. It can be shown that for any two starting numbers, this repeated reduction eventually produces a pair where the second number is 0. Then the GCD is the other number in the pair.

   Write a method named `gcd` that computes the greatest common divisor of two integers. Add some code to `main` that tests the new method.

7. Ackerman's function, $A(m, n)$ is defined:

$$\begin{aligned} A(0,n) &= n+1 && \text{for } n \geq 0 \\ A(m,0) &= A(m-1,1) && \text{for } m > 0 \\ A(m,n) &= A(m-1, A(m,n-1)) && \text{for } n \geq 0 \end{aligned} \tag{3}$$

   Write a method named `ack` that evaluates Ackerman's function. Use your method to evaluate `ack(3, 4)`, which should be 125. What happens for larger values of `m` and `n`?

A little String manipulation

1. Download, compile and run:

   `http://rocky.wellesley.edu/cs230/code/hw00/Palindrome.java`

   If you read the code, you will see that it reads through all the words in `/usr/share/dict/words` and prints any words for which `isPalindrome` returns true.

   A palindrome is a word that reads the same both forward and backward, like "otto" and "palindromeemordnilap." An alternative, recursive definition of a palindrome is "either an empty string or a word whose first letter is the same as the last, and whose middle is a palindrome."

   Fill in the body of `isPalindrome` so that it returns true if the parameter is a palindrome. When you compile and run the program, it should print all the palindromes in the dictionary.

   It would be an excellent idea to write helper methods called `first` (which returns the first letter of a String), `last` (which returns the last letter of a String), and `middle` (which uses `subString` and returns all but the first and last characters, or an empty string if there are none).

   WARNING: The output of this program contains several words of an adult nature. Viewer discretion is advised!

Bonus Question: Making Change

In an old Saturday Night Live, there was a fake ad for a bank that specialized in making change. That's all they did, but they were very good at it.

You have been hired by this fictitious bank to write a program for them that counts the number of possible ways to make change for a given amount of money. Let's assume that we are dealing with coins only and not concern ourselves with paper money.

Ultimately, you want a method that takes three arguments:

- The number of cents, as an integer.
- An array of integers named `coinage` that specifies the denominations of the coins that can be used. For American currency, this array would contain the values `1 5 10 25 50`.
- How many different kinds of coins can be used in the solution. For example, if this value is 4, then only the first four denominations, `1 5 10 25`, can be used.

Thus the invocation `countChange (100, coinage, 5)` would ask how many ways there are make change for a dollar, using pennies, nickels, dimes and quarters and half-dollars. The answer is 292.

1. Have I given you enough information to solve this problem or is there more that needs to be specified? If the latter, then feel free to disambiguate in whatever way makes the implementation easiest.
2. Design an algorithm for doing this computation.
3. Implement it. Please call it `countChange` and put it in `Recurse.java`.

WARNING: This question is hard. I want you to think about it because it is worth thinking about. Even if you don't come up with an optimal answer, the process of thinking about it will be valuable.

1 What to turn in

The following are the steps you should follow for turning in assignments throughout the semester.

1. Review your code and make sure it is indented properly. Remove any unused methods or unreachable code. Remove code that is commented out.
2. Review the names of the variables and methods to make sure that they are meaningful (with the exception of things like `i` and `x` that are conventional). If the role of a variable is not obvious, add a comment that explains it.
3. Add a comment at the beginning of the program that contains your name, email address, and which homework it is.
4. Add a comment before each method that explains what it does, at an appropriate level of abstraction. Document any unexpected behavior the method might have.
5. Within each method add comments that document anything unusual or non-intuitive. Do not write comments that are redundant with the code.

6. Arrange the code and comments visually in a way that makes the program easy to read. Avoid line breaks that make the layout of the code unpleasant. Your program should fit into an 80-column screen without wrapping any lines.
7. Print the code in a comfortable-sized font (12 point), portrait aspect. The best way to do that is

   ```
   a2ps -1 Filename.java
   ```

8. On the hardcopy, write your name in the upper right-hand corner and identify which assignment and which part of the assignment it is.
9. Highlight the name of each method, preferably using a highlighter, but optionally underlining with a non-black pen.
10. If the assignment includes code that you did not write, put a box around it and cross-hatch the box.
11. Arrange the parts of the assignment in the order they were presented. Staple all pages of the assignment together in the upper left-hand corner.

If you follow these instructions, you make it easier for us to grade your work and provide prompt useful feedback. Thank you!

Credits

`power`, `morePower` and `ack` are from Standish, *Data Structures in Java*, Addison Wesley.

The GCD problem and the coin-counting problem are from Abelson and Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, 1987.