Computer Science CS136 (Fall 2021)
Duane Bailey & Aaron Williams
Laboratory 2
Recursion

Objective. Practice solving problems using recursion.

Discussion. This week's lab is structured as several small problems that can be solved in isolation. Recursion can be a difficult concept to master and one that is worth concentration on separately before using it in large programs. Recursive solutions can often be formulated in just a few concise, elegant lines, but they can be very subtle and hard to get right.

We would like you to write this week's problem solutions using recursion. Take time to figure out how each problem is self-referential in nature and how you could formulate the solution to the problem if you already had the solution to a smaller, simpler version of the problem. You will often need to have faith that your solution will ultimately be correct, even early in the design of the code. Remember, recursion requires three things: one or more base cases, a reduction to a subproblem, and a little progress that glues the solutions to the subproblem together. If you learn to think recursively, the solutions to many problems will seem very intuitive.

This Week's Tasks. Here is what we expect you to do, this week:

1. Clone your lab2 repository:

   ```
   cd ~/cs136
   git clone https://evolene.cs.williams.edu/cs136-labs/22xyz3/lab2.git
   ```

   replacing 22xyz3 with your CS username. This will create a subdirectory, `lab2` in your `cs136` directory.

2. Change to the `lab2` directory and look around:

   ```
   cd ~/cs136/lab2
   ls
   ```

   There is one file we expect you to modify: `Recursion.java`. This contains the declarations for a number of recursive procedures. You are not allowed to change the declarations. Your job is to complete each method, implementing it *using recursion*. Some methods may require a protected recursive "helper method" that does most of the work. For each method `xyz`, we would also like you to write a *protected* test method, `testXyz`, that uses the structure package's `Assert` class to test enough cases to assure yourself that the `xyz` method does what it is supposed to. Do not limit yourself to the examples you see in this handout; aim to test the simplest or most instructive cases where possible. Here's how we might use `Assert` to test to see if we're computing `binary(5)` correctly:

   ```
   protected static void testBinary() {
       Assert.condition(binary(5).equals("101"),
                        "The binary representation of 5 should be 101.");
       // more testing, here...
   }
   ```

The message is what is printed if the assertion fails. As you finish off each problem, place a call to your test method in the main method of the recursion class. In this way, when you run the recursion class, the tests should silently all work (and the program should appear to do nothing).

3. As an example of how we'd like to proceed, we've implemented triangular(n), the computation of the sum $1 + 2 + \cdots + n$. To check our work, we've implemented testTriangular(), which silently tests several cases. Later, in the main method, we call testTriangular() to make sure that we haven't introduced any errors, and that we don't in the future, either.

4. Write a method, tetrahedral(n), that computes the number of cannonballs that are stacked in a tetrahedral pyramid with n layers. The top has one cannonball, which rests on a layer with $1+2 = 3$ cannonballs. This layer sits atop a layer with $1+2+3 = 6$ cannonballs. The pattern continues. Here are some example values:

method call	**result**
tetrahedral(0)	0
tetrahedral(1)	1
tetrahedral(2)	4
tetrahedral(3)	10

5. Write a method, String parensFrom(String str), that returns a string of the paren-like characters ("()<>[]{}") in the order they appear in str. Here is what we're looking for:

method call	**result**
parensFrom("")	""
parensFrom("Hello, world.")	""
parensFrom("(^_^)")	"()"
parensFrom("array[i] = (array[i]+1)*2")	"[]([])"
parensFrom("1.) Add 1 cup [organic] flour")	")[]"
parensFrom(") [> { } <] (")	")[>{}<]("

6. Write a method, boolean isBalanced(String s), that takes a string, s, of round parentheses, square brackets, angle brackets, and curly brackets. The method should return true exactly when the various symbols are balanced. A string is balanced if

   (a) The string is empty, or

   (b) The string contains a pair of adjacent matching brackets as a substring and remains balanced when that substring is removed.

   Here is a table of possible strings and whether they are balanced or not:

method call	**result**
isBalanced("(){[]}")	true
isBalanced("{[]}")	true
isBalanced("{}")	true
isBalanced("")	true
isBalanced("[(])")	false
isBalanced(parensFrom("{ println(a[3]); println(); }"))	true

7. Write a function, `binary(n)`, that takes a non-negative integer, `n`, and returns a string that gives the base-2 (binary) representation of `n`. If `n` is even, its rightmost digit is 0; when it is odd, the rightmost digit is 1. Here are some examples:

method call	**result**
`binary(0)`	`"0"`
`binary(1)`	`"1"`
`binary(2)`	`"10"`
`binary(5)`	`"101"`
`binary(10)`	`"1010"`
`binary(11)`	`"1011"`

Notice also that removing the rightmost binary digit effectively divides the number represented by two. Your recursive method will want to make use of these facts.

If you've made it this far, you've likely earned most of the credit for the lab. Congrats! The methods that follow are a bit more complex and will require more thought to implement.

8. A *subsequence* of charactacters found in a string, `str`, is any string that can be constructed by possibly deleting characters from `str`. Write a method, `subsequences(str)` that collects all the subsequences of `str` in a `Vector<String>`. Here are the subsequences of `"Eve"`:

```
"", "E", "v", "e", "Ev", "Ee", "ve", "Eve"
```

Notice there are $2 \times 2 \times 2 = 8$ subsequences because half the subseqeneces include each letter and the other half do not. Also, notice that if there are duplicate letters in the string, then we will have duplicate subsequences; that's fine. For example, the subsequences of `"eve"` include `"e"` twice. We suggest that you center your recursion in a helper method,

```
protected static void subsequencesHelper(String str, String soFar,
                                         Vector<String> result)
```

This method recursively constructs a sequence from `str`, accumulating the characters it intends on keeping in a string `soFar`, and, when complete, adds them to `result`. In processing `str`, we

(a) build all subseqeunces containing the first character (by appending it to `soFar` in a recursive call), and then

(b) build all subsequences not containing the first character (by leaving `soFar` unchanged in a recursive call).

Here are some examples of using `subsequences`:

method call	**result**
`subsequences("Bill").size()`	16
`subsequences("Williams").contains("ills")`	true
`subsequences("Bailey").contains("ale")`	true

9. Write a method, `canSumTo(int[] nums, int n, int target)`, that takes an array with `n` integers, and return `true` exactly when some subsequence of the first `n` integers in `nums` sums to `target`. Since this method returns a boolean, it's not important to explore all subseqeunces: once you find *some* selection of `nums` that sums to `target`, you can return `true`. If no selection of `nums` sums to target, return `false`.

   Here are some examples of using `canSumTo`. Suppose `vals` contains `{1,2,3,-7}`. Then we have the following results:

method call	**result**
`canSumTo(vals,4,0)`	`true`
`canSumTo(vals,4,4)`	`true`
`canSumTo(vals,4,-6)`	`true`
`canSumTo(vals,4,13)`	`false`

10. Before you turn in your work, make sure that you have completed all the methods required, including the protected test methods. The main method of your `Recursion` class should call the test methods, and should be silent when run. Make sure your code is well written, and documented appropriately.

Submitting Your Work. To get credit for this week's lab make sure that you've added, committed, and pushed the files `Recursion.java` and `honorcode.txt` (containing your signature). Recall that `git status` and `git push` should suggest that everything is up-to-date.

⋆