Lab 2: Recursion

Due 11:59 pm Mon, Feb 25, 2008

1 Assignment

Write a single class stored in Recursion.java that provides the following public static methods:

1. *Examples discussed in lab and lectures:*
    a. public static int sumDigits(int n)
    b. public static int countCannonballs(int height)
    c. public static boolean isPalindrome(String str)
    d. public static boolean isBalanced(String str)
2. *New examples:*
    a. public static void generateSubstrings(String str, ArrayList<String> result)
    b. public static void generatePermutations(String str, ArrayList<String> result)
    c. public static String toBinary(int number)
    d. public static boolean subsetSum(int[] sets, int targetSum, ArrayList<Integer> result)

as well as a series of static test routines that verify the correctness of your implementations. These methods are described below.

In addition, provide an empty main method. You may use as many helper classes and methods as you need. Ensure that helper methods are all private. **All public methods must be implemented using** (non-trivial) **recursion**, even where an alternative iterative implementation is possible. This means that unless noted ("you can use iteration"), you may not use FOR or WHILE loops. The point of this lab is to prepare you for the recursive data structures and algorithms in subsequent labs.

When complete, clean up your directory (see the clean136 script) and then submit your solution using the command:

```
turnin –c 136 Recursion.java
```

For this lab, your solution should be a single file named "Recursion.java". Do not submit files ending in ".class" or "~".

1.1 sumDigits

Method sumDigits takes a non-negative integer and returns the sum of its base-10 digits. For example, sumDigits (1234) returns $1 + 2 + 3 + 4 = 10$. Note that it is easy to split an integer into two smaller pieces by dividing by 10 (e.g., 1234/10 = 123, and 1234 % 10 = 4).

1.2 countCannonballs

Spherical objects, such as cannonballs, can be stacked to form a pyramid with one cannonball at the top, sitting on top of a square composed of four cannonballs, sitting on top of a square composed of nine cannonballs, and so forth. Write a recursive method countCannonballs that takes as its argument the height (number of layers) of a pyramid of cannonballs and returns the total number of cannonballs that the stack contains.

1.3 isPalindrome

Write a recursive method isPalindrome that takes a String and returns true if that String is identical to itself reversed. For example,

```
isPalindrome("mom")      returns true
isPalindrome("cat")      returns false
isPalindrome("level")    returns true
```

You may assume that the input string contains no spaces. A reasonable solution uses substring; a more efficient one uses a helper method to avoid substring (which causes memory allocation).

1.4 isBalanced

In the syntax of most programming languages, there are characters that occur only in nested pairs, called bracketing operators. Java, for example, uses ( ), [ ], and {} as bracketing operators. In a properly formed program, these characters will be properly nested and matched. To determine whether this condition holds for a particular program, you can ignore all the other characters and look simply at the pattern formed by the parentheses, brackets, and braces. In a legal configuration, all the operators match up correctly, as shown in the following example:

```
{ ( [ ] ) ( [ ( ) ] ) }
```

The following configurations, however, are illegal for the reasons stated:

(([ ])	The line is missing a close parenthesis.
) (	The close parenthesis comes before the open parenthesis.
{ (})	The parentheses and braces are improperly nested.

Write a recursive method that takes a String named str from which all characters except the bracketing operators have been removed. The method should return true if the bracketing operators in str are balanced, which means that they are correctly nested and aligned. If the string is not balanced, the method returns false. Your method may use a for loop with a fixed number of iterations, but the main process of the solution should be recursive.

Although there are many other ways to implement this operation, you should code your solution so that it embodies the recursive insight that a string consisting only of bracketing characters is balanced if and only if:

1. The string is empty, or
2. The string contains "( )", "[ ]", or "{}" as a substring **and** remains balanced if you remove that substring.

For example, the string "[( ){}]" is shown to be balanced by the following chain of calls:

```
isBalanced("[( ){}]")
    … isBalanced("[{}]")
        … isBalanced("[]")
            … isBalanced("")
```

Don't try to implement this one to avoid memory allocation; it is not possible by this algorithm (although there are other algorithms that can succeed!)

1.5 generateSubstrings

Write a method that adds all subsets of the letters in its first argument, str, to the ArrayList that is its second argument. For example,

```
generateSubstrings("ABC", result)
```

adds to result the strings:

```
"", "A", "B", "C", "AB", "AC", "BC", "ABC"
```

The order of the strings does not matter. In your test code, you will probably want a helper method that prints all of the strings in an ArrayList<String>). You may use iteration in any way you see fit for this problem. Do not worry too much about eliminating all memory allocation in this method.

1.6 generatePermutations

Write a method that adds out all permutations of the letters in a string argument to the ArrayList<String> passed as the second argument. For example,

generatePermutations ("ABC", result)

adds to result the strings:

"ABC", "ACB", "BAC", "BCA", "CAB", "CBA"

The order in which they are added does not matter. If a letter appears twice in the input string, then it is acceptable to produce some duplicate permutations in the output that occur because they are different permutations of the same letter. Otherwise there should be no duplicates.

You may find it very useful to write a helper method

public static void permuteHelper(String soFar, String rest, ArrayList<String> result)

that is initially called as substringHelper("", str, result). The variable soFar is a fixed prefix to be included as part of the output, and rest is a suffix whose characters must still be permuted. Thus, if rest is empty, then soFar will be a complete permutation. If rest is not empty, then you must extend the prefix soFar by each possible choice of letter remaining in rest, and recursively compute all permutations of the remaining letters. You may use iteration in any way that you see fit in your solution.

1.7 toBinary

Computers represent integers as sequences of bits. A bit is a single digit in the binary number system and can therefore have only the value 0 or 1. The table below shows the first few integers represented in binary:

Binary	Decimal
0	0
1	1
1 0	2
1 1	3
1 0 0	4
1 0 1	5
1 1 0	6

Each entry in the binary side of the table is written in its standard binary representation, in which each bit position counts for twice as much as the position to its right.

Basically, this is a base-2 number system instead of the decimal (base-10) system with which most people are familiar. Write a recursive method that returns a string containing the binary representation for a given integer. You must not have leading zeros on your output, although the integer 0 must translate to "0". Note that order is very important for this problem!

Hint 1: Use Google to search for "6 in binary" and you'll find a good way of testing your program.
Hint 2: x >> 1 is identical to x / 2, which is identical to saying "shift the bits of this integer one place to the right"
Hint 3: x & 1 is identical to x % 2, which is identical to saying "tell me the last bit of this integer"

1.8 subsetSum

Subset Sum is an important and classic problem in computer theory that is, given a set of integers and a target number, to find a subset of those numbers that exactly sums to the target number. For example, given the set {3, 7, 1, 8, -3} and the target sum 4, the subset {3, 1} sums to 4. A subset means that you can't reuse a number (although if, e.g., the number appears twice in the input set, then you *can* use it twice).

Implement a solution to the Subset Sum problem. Your method should fill out the result argument with a subset, if one exists, or leave it empty if one does not exist. Return true if a subset was found and false otherwise. Draw inspiration from your generatesSubsets code. The following method may be helpful:

```
/** Copies all but the element at removeIndex from src to dst. */
private static void removeOne(int[] src, int[] dst, int removeIndex) {
   assert src.length == dst.length + 1;
   // Copy the first part
   System.arraycopy(src, 0, dst, 0, removeIndex);

   // Copy the rest
   System.arraycopy(src, removeIndex + 1, dst, removeIndex, src.length - removeIndex - 1);
}
```

2 Evaluation

The grading guidelines for this assignment are below. Note that this week, correctness counts for a lot more than usual; that's because the design is mostly set out for you by the specification. Although they have varying difficulty, each sub-part of the assignment has equal weight except for subsetSum, which counts for 2× as much as the others.

Correctness	**20**
Readability	**10**
Design and Efficiency (and Elegance)	**10**
Total	**40**

3 Advice

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 concentrating on separately before using it in large programs. Each problem will have a fairly short solution, but that doesn't mean you should put this assignment off until the last minute though. Recursive solutions can often be formulated in just a few concise, elegant lines but they can be very subtle and hard to get right.

Recursion is a tricky topic so don't be dismayed if you can't immediately sit down and code these perfectly the first time. Take time to figure out how each problem is recursive 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 same problem. You will need to depend on a recursive "leap of faith" to write the solution in terms of a problem you haven't solved yet. Be sure to take care of your base case(s) lest you end up in infinite recursion.

The great thing about recursion is that once you learn to think recursively, recursive solutions to problems seem very intuitive. Spend some time on these problems and you'll be much better prepared when you are faced with more sophisticated recursive problems.

The approximate line counts for my solutions are below (including comments, whitespace, and assertions). Yours should probably be within a factor of two of mine in length.

sumDigits	10
countCannonballs	20
isPalindrome	20 (inefficient) 30 (efficient)
isBalanced	30
generateSubstrings	11
generatePermutations	30
toBinary	35
subsetSum	45 (including removeOne helper)