Lab 3 **Mouna Kacimi, Werner Nutt, Camilo Thorne**

3. Recursive Programs for Integers

This is a set of exercises to write recursive programs over the integers. For each program, first develop a solution in pseudo-code. Then code in Java. The results should be printed to standard output.

1. Greatest common divisor

Write a function with the following (Java) signature:

```
public static int gcd(int n, int m),
```

that returns the greatest common divisor *gcd*(n, m) of two positive integers n and m. For instance, if $n = 21$ and $m = 11$, your function should return 1; if $m = 49$, then it should return 7.

1. First, write a recursive version `gcdRec`. Start with pseudo-code, then refine the pseudo-code to the Java version.

   Hints: Note that $\mathit{gcd}(n, m) = \mathit{gcd}(n \mod m, m)$ if $n > m$. What is the base case of your recursion?
   For the Java implementation, write a method that calls `gcdRec` and prints the result of the call to standard output.

2. Turn the recursive version into an iterative version `gcdIter` with a while-loop. As before, first write pseudocode, then Java code.

2. Binary representation of integers

Write a procedure with the following (Java) signature:

```
public static void decToBin(int n).
```

The procedure takes a positive integer n (in decimal) as input and returns its binary representation $Bin(n)$, printing the bits in the correct order. For instance, if $n = 10$ it should print 1010.
Hint: Repeatedly divide n by 2 and print the remainders in reverse order, that is, print the remainder of the first division as the last digit.

1. First, write a recursive version `decToBinRec`. Start with pseudo-code, then refine the pseudo-code to the Java version.

   Hint: Use the following method: Repeatedly divide n by 2 and print the remainders in reverse order, that is, print the remainder of the first division as the last digit.

2. Turn the recursive version into an iterative version `decToBinIter`. As before, first write pseudocode, then Java code.

   Hint: There is a difficulty with the iterative version that does not show up in the recursive version: you first compute the least significant digit, but you have to print it last. To remember the bits that you have computed, but cannot print yet, you need an additional data structure that was not necessary in the recursive version.

2. Subsets of a set

In this task, we will consider how to recursively compute all non-empty subsets of some finite set. The finite set is represented as an array A of n characters. As before, the result should be printed on standard output.

1. Write a recursive Java method of signature

   ```
   public static void recComb(char[] A)
   ```

   that computes the $2^n - 1$ nonempty subsets of A. If A consists of characters a, b, and c it should return, e.g.,

   $abc \quad ab \quad ac \quad a \quad bc \quad b \quad c$

   (the order does not matter).

2. Modify program 1. so that it prints its results in *lexicographic order*.

Hint: Suppose your set A consists of $\{a, b, c, d\}$. To print the nonempty subsets of A, print first the subsets that contain a and then those that do not contain a. (Note that this can be achieved by *two* recursive calls.) To print the subsets that contain a, print first those that contain b and then those that do not contain b. And so on.
To print the subsets that do not contain a, print first the ones that contain b and then those that do not contain b. And so on. Finally, to print the subsets containing, say, a, c, d, just print a, c, d.
Generalise a procedure from this idea.
The examples above show that when making a recursive call, you need to remember which are the characters you will print for *all* sets covered by that call. To remember them, combine them into a string and pass the string as an argument.