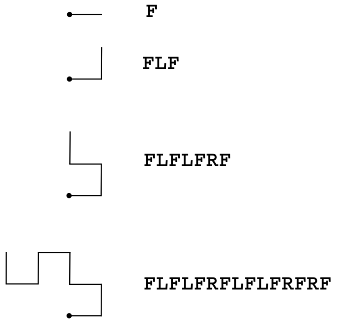
Section Readings: 1.5, 1.6, 2.1, 2.2

Problem 1. (*Dragon Curves*) Write a program `Dragon` that takes an `int` value N as command-line argument and prints the instructions for drawing the dragon curve of order N. The instructions are strings of F, L, and R characters, where F means "draw line while moving 1 unit forward," L means "turn left," and R means "turn right." A dragon curve of order 0 is just the character F, and a curve of order N is a curve of order $N-1$ followed by an L followed by a curve of order $N-1$ traversed in reverse order with R replaced by L and L replaced by R.

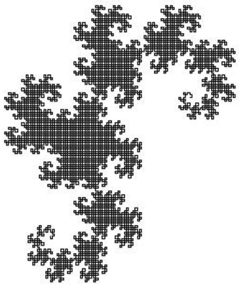
```
$ java Dragon 0
F
$ java Dragon 1
FLF
$ java Dragon 2
FLFLFRF
$ java Dragon 3
FLFLFRFLFLFRFRF
```

Dragon curves of orders 0, 1, 2, and 3 and the corresponding instructions are shown below.



Problem 2. (*Visualizing Dragon Curves*) Write a program `DragonPlot` that reads from standard input the instructions produced by `Dragon` (from Problem 1) for drawing a dragon curve, generates a drawing of the curve (shown below), and saves it in a file called `dragon.jpg`.

```
$ java Dragon 13 | java DragonPlot
```



Problem 3. (*Numbers Library*) Implement `Numbers`, a library of static methods with the following API:

```
public class Numbers

    // Return true if n is prime, and false otherwise.
    public static boolean isPrime(int n)

    // Return the greatest common divisor of a and b.
    public static int gcd(int a, int b)

    // Return true if a and b are coprime, ie, their gcd is 1, and false
    // otherwise.
    public static boolean coprime(int a, int b)

    // Return the sum of the proper divisors of n. Eg, if n is 6, return
    // 1 + 2 + 3 = 6, since 1, 2, and 3 are the proper divisors of 6.
    public static int sumOfProperDivisors(int n)

    // Return true if n is perfect, ie, its proper divisors add up to n,
    // and false otherwise.
    public static boolean isPerfect(int n)

    // Return true if a and b are amicable, ie, the proper divisors of a
    // add up to b and the proper divisors of b add up to a.
    public static boolean amicable(int a, int b)
```

```
$ java Numbers
false
true
8
true
true
false
true
```

Problem 4. (*Counting Primes*) Write a client program `PrimeCounter` that takes an `int` value n as command-line argument and prints the number of primes less than or equal to n. Your implementation must use the `isPrime()` method from `Numbers` to check if a number is prime or not.

```
$ java PrimeCounter 100
25
$ java PrimeCounter 10000000
664579
```

Problem 5. (*Counting Totatives*) Write a client program `TotativeCounter` that takes an `int` value n as command-line argument and prints the number totatives of n, ie, the number of positive integers less than or equal to n that are coprime to n. Your implementation must use the `coprime()` method from `Numbers` to check if two numbers are coprime.

```
$ java TotativeCounter 100
40
```

Problem 6. (*Coprime Pattern*) Write a client program `CoprimePattern` that takes an `int` value n as a command-line argument and prints an n-by-n matrix such that the element in row i and column j ($1 \leq i, j \leq n$) is a `"*"` (a star) if i and j are coprime and a `" "` (a space) otherwise. The row numbers should be printed at the end of each row. Your implementation must use the `coprime()` method from `Numbers` to check if two numbers are coprime.

```
$ java CoprimePattern 10
* * * * * * * * * * 1
*   *   *   *   *   2
* *   * *   * *   * 3
*   *   *   *   *   4
* * * *   * * * *   5
*       *   *       6
* * * * * *   * * * 7
*   *   *   *   *   8
* *   * *   * *   * 9
*   *       *   *   10
```

Problem 7. (*Perfect Numbers*) Write a client program `PerfectNumbers` that takes an `int` value n as command-line argument and prints the numbers less than or equal to n that are perfect. Your implementation must use the `isPerfect()` method from `Numbers` to check if a number is perfect.

```
$ java PerfectNumbers 10000
6
28
496
8128
```

Problem 8. (*Amicable Pairs*) Write a client program `AmicablePairs` that takes an `int` value n as command-line argument and prints distinct pairs (i, j) with $i \neq j$ and $1 \leq$

$i, j \leq n$, such that i and j are amicable. Your implementation must use the `amicable()` method from `Numbers` to check if two numbers are amicable.

```
$ java AmicablePairs 3000
(220, 284)
(1184, 1210)
(2620, 2924)
```

Files to Submit:

1. `Dragon.java`
2. `DragonPlot.java`
3. `Numbers.java`
4. `PrimeCounter.java`
5. `TotativeCounter.java`
6. `CoprimePattern.java`
7. `PerfectNumbers.java`
8. `AmicablePairs.java`
9. `report.txt`