Exercises 2

In the following exercises, **make sure to rely on thread management facilities discussed in the lecture**. Our goal is to gain confidence with the basics; we'll deal with more advanced tools later.

Task 2.1. Finding Palindromes

Files `words1.txt` and `words2.txt` consist of English words, one word per line. I want to print out all *palindromes* found in these files (a palindrome is a word that reads the same backward as forward, for example, *radar*, *kayak* or *racecar*). Each identified palindrome should be printed only once.

To do it, I wrote the following two-threaded program:

```
import java.util.*;
import java.io.*;

class Global
{
    public static Vector palindromes = new Vector();
}

class MyThread implements Runnable
{
    private String filename;

    public MyThread(String filename_) { filename = filename_; }

    public void run()
    {
        try
        {
            BufferedReader reader = new BufferedReader(new FileReader(filename));
            String line;
            while ((line = reader.readLine()) != null) // read file line by line
            {
                // reverse the next line
                String revline = (new StringBuilder(line)).reverse().toString();

                // if a new palindrome is found
                if(line.equals(revline) && !Global.palindromes.contains(line))
                {
                    System.out.println(line);
                    Global.palindromes.add(line);
                }
            }
        }
        catch(IOException e)
        {
        }
    }
}
```

```
class Palindromes
{
    static public void main(final String args[])
    {
        new Thread(new MyThread("words1.txt")).start();
        new Thread(new MyThread("words2.txt")).start();
    }
}
```

Can this program provide incorrect output? If yes, explain why.

Task 2.2. State Space Diagram v2

Draw a state space diagram for the following concurrent program:

MEMORY	`A = 0`
Task 1	`A=2; A++;`
Task 2	`A=3; A--;`

Assume that all operations in this code are atomic.

Task 2.3. Crack the Password v2

Class ex02_fileCrypto implements the following methods:

```
class ex02_fileCrypto {
    // encrypt the given file with the given password,
    // and store the resulting data in the specified output file
    // (example: Encrypt("file.txt", "file.encrypted", "pass12345"))
    static public void Encrypt(String inFileName,
                String outFileName, String password) throws Exception
    { ... }

    // decrypt the given file using the given password,
    // and store the resulting data in the specified output file
    // (example: Decrypt("file.encrypted", "file.txt", "pass12345"))
    // in case of success, return true;
    // in case of failure (due to incorrect password), return false
    static public boolean Decrypt(String inFileName,
                String outFileName, String password) throws Exception
    { ... }
}
```

Files ex02_longtail.enc, ex02_pride.enc, and ex02_scarlet.enc were encrypted using `Encrypt()` method, and, unfortunately, I cannot decrypt them because the passwords are lost. I only remember that each password was at most five characters long and contained digits only.

Write a program that uses brute force to recover the password (in other words, the program tries all possible combinations of 1 to 5 digits, until the password is

discovered). The program should speed up the work by running four parallel processes. Each process takes a quarter of the overall work. When the password is found (it might take several minutes), report it. Please submit both the program source and the passwords you found.

Task 2.4. Armstrong Numbers

A positive *m*-digit integer is called an *Armstrong number* if it is equal to the sum of the *m*-th powers of its digits. For example, $153 = 1^3 + 5^3 + 3^3$, and $1634 = 1^4 + 6^4 + 3^4 + 4^4$.

All one-digit numbers are Armstrong numbers, and the first five Armstrong numbers larger than 9 are 153, 370, 371, 407 and 1634.

Your task is to write a program that prints (in any order) all Armstrong numbers n the range [10, *N*), where the value of *N* is provided by the user. The program should divide the work between *P* threads that analyze different parts of the input range. The value of *P* is also provided by the user.

Example input data: `10000 4`

Example output data: `153 370 371 407 1634 8208 9474`