Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Übungen zur Vorlesung Informatik II (D-BAUG) FS 2017

D. Sidler, F. Friedrich

http://lec.inf.ethz.ch/baug/informatik2/2017

Problem set # 3 **6.3.2017 – 14.3.2017**

Problem 3.1. Reverse Words

Open the task description here: https://codeboard.ethz.ch/inf2baugex03t01. In this task you write a program that takes strings as an input and reverses the order of the character in the strings. For instance for the input string `Hello` the program returns the string `olleH`.

Looking at the `main` function in Main.java, you can see that it reads a string from the input and then it checks if this string is equal to "exit". If this is the case the program prints out "exit." and terminates. Otherwise the program is going to reverse the characters in the string and output it. After reversing a string the program goes back to the beginning of the `while` loop and reads in the next string. This means the program will read in new string until you enter "exit".

```
Scanner input = new Scanner (System.in);
while (true) {
    String str = input.next();
    if (str.equals("exit")) {
        System.out.println("exit.");
        break;
    }
    //TODO reverse the input string str and print it out
}
```

Implement the functionality to reverse the string, you can make use of the java function `charAt(int pos)` to get the character at position `pos` in a string.

You can test your program by un-commenting the annotation `@RunTests`. Once you pass the test you can submit your program.

Problem 3.2. Matrix Multiplication

Open the task description here: https://codeboard.ethz.ch/inf2baugex03t02. In this task you are implementing two functions, the first one `readMatrix` reads a matrix from the input, the second one `multiplyMatrix` multiplies two matrices and returns the resulting matrix.

First have a look at the function `readMatrix` in the file Main.java. In order to read a matrix from the input use the Scanner. The format of a $r \times c$ matrix is explained with the following 2×3 example matrix:

```
2 3
0.10000 0.20000 0.30000
1.10000 1.20000 1.30000
```

The first line contains the number r of rows followed by the number c of columns, both integers. The following r lines contain c doubles each, representing a $r \times c$ matrix. The numbers are separated by whitespaces.

After implementing this function you can complete the implementation of the `multiplyMatrix` function. The goal is to do a matrix multiplication ($m1 \times m2$), where $m1$ has size $(N1 \times M1)$ and $m2$ has size $(N2 \times M2)$ and $N1, M1, N2, M2 > 0$. The resulting matrix should be of size $(N1 \times M2)$. Do not modify the input data.

After implementing the function you can test your program by un-commenting the annotation `@RunTests`. Once you pass the test you can submit your program.

Problem 3.3. Artificial Words

Open the task description here: https://codeboard.ethz.ch/inf2baugex03t03.

Goal of this task is to generate artificial words from an alphabet. We consider the alphabet $\Omega = \{'\_', 'A', \dots, 'Z'\}$. '_' stands for the empty letter.

From a very long text, for each letter $x \in \Omega$ the relative frequency of all potentially following letters $y \in \Omega$ is determined as P_{xy}. This yields a probability matrix $(P_{xy})_{x,y\in\Omega}$. Thus, P contains the relative frequencies for character combinations of the form "AA", "ST", or "_A" (empty letter before: word starts) or "Z_" (empty letter after: word ends).

Now, from the probability matrix P a word can be generated by starting with the empty letter, $x = '\_'$, and subsequently drawing new characters from $P_{x\cdot}$ until the empty letter appears again.

In the following you see an excerpt of a probability matrix P, generated from a tale from Johann Wolfgang von Goethe and some generated words.

	'_'	'A'	'B'	'C'	'D'	'E'	'F'	'G'	'H'	···
'_'	0.000	0.073	0.040	0.000	0.154	0.063	0.028	0.047	0.045	···
'A'	0.008	0.002	0.049	0.048	0.006	0.000	0.011	0.047	0.031	···
'B'	0.061	0.061	0.000	0.001	0.000	0.536	0.001	0.015	0.004	···
'C'	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0902	···
'D'	0.208	0.088	0.000	0.004	0.000	0.431	0.000	0.000	0.000	···
'E'	0.240	0.000	0.014	0.005	0.012	0.001	0.009	0.015	0.022	···
'F'	0.204	0.052	0.000	0.000	0.000	0.167	0.045	0.014	0.039	···
'G'	0.174	0.027	0.000	0.000	0.002	0.514	0.003	0.001	0.000	···
'H'	0.222	0.078	0.001	0.001	0.002	0.165	0.002	0.001	0.003	···
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

```
WOPT DEDERUTERM SCHRCHLAGEGOR DIN ASTE FOSITEMER IGTES AUF SENNER DEIMER EMAR ZTENNALENDIN
WAHWAUR VOCHENS WRINGENE DESTRN ZERERTER ALISCHRDERCH IGSSONSCH VOLIGET ALZUNSCHR
LDULE AUNN UN WABGEIE D BR KENUNEIGT SPTCHIHMERR OLICHN DE SORBEN UESERER
```

In the file Main.java you see three functions: `Sample`, `GenerateWord`, and `GetMatrix`. You have to complete the first two functions. The `Sample` function returns the index of the next letter where the index is in the range 0-26. In the function `GenerateWord` you only need to complete one line. The function `GetMatrix` is already implemented. It reads the probability matrix from the file "probabilities.txt". Do not change this function nor the probabilities in the file.

Once you have implemented the two functions, you can test your program by un-commenting the annotation `@RunTests`. Once you pass the test you can submit your program.