Chapter 1: Set one: Introduction.
Deadline: Friday, September 20, 23:59 (11:59 PM)

Please consult the 'Practical assignments instructions and rules' document before proceeding with the exercises.

Exercise 1. [1 point]
Purpose of this exercise: familiarization with an acceptable use policy.

Download (and submit a link in the report to) the University of Groningen's Acceptable Use Policy (AUP). In the report, provide your answers to the following questions:

1. What (if any) are the differences between the responsibilities of 'ordinary' users and systems managers? Do systems managers have special privileges and responsibilities (if so, what are they)?
2. What is the ground-rule upon which the RUG's AUP is based;
3. Mention four advices for users of 'RuGnet';
4. Describe four actions that are prohibited by the RUG's AUP;
5. What sanctions can be applied to those who violate the AUP?
6. If a sanction is applied to you, where can you go to challenge that sanction?

Exercise 2. [3 points]
Purpose of this exercise: construct a tool to work with a substitution cipher.

Write a program that, given a query consisting of multiple substitution cipher encryption and decryption requests, consecutively applies them to a given plaintext/ciphertext and outputs the final result. Each request has the following structure:

```
(d | e) (mapping | shift)
```

Where mapping represents a permutation of the 26 character English alphabet, and shift is a signed 4 byte integer ($-2^{31} \leq$ `shift` $\leq 2^{31} - 1$). A decryption request should revert the specified mapping or shift on the given text.
Note: an encryption with a shift divisible by 26 does not modify the text and a shift equal to 3 is equivalent to the classical Caesar encryption. The correct sequence for calling the substitution cipher is the following:

1. Call the tool on the command line without arguments.
2. On the first stdin line, indicate the series of requests (following the above structure), followed by the newline character.

3. On the following lines, insert the text upon which the query will be applied. The first request will use this text to produce the ciphertext/plaintext which will be in turn used by the second request, and so forth until the last request.

As an example, the following string could be inserted on the first line of stdin:

```
d 5 e 12 d zyxwvutsrqponmlkjihgfedcba e 3
```

The substitution cipher tool then applies encryption/decryption in the following order:

1. Decrypt with a shift of 5;
2. Encrypt with a shift of 12;
3. Decrypt with the map zyxwvutsrqponmlkjihgfedcba;
4. Encrypt with a shift of 3.

Example: Assume that we attempt to encrypt the following text:

```
Hello, I am studying Information Security!
```

... using the following encryption/decryption sequence:

```
e 5 d 4 e zyxwvutsrqponmlkjihgfedcba
```

Your program should produce the following output:

```
Runnk, Q ym gfevaqls Qltkhmyfqkl Guwehqfa!
```

Notice that non-character symbols remain unchanged (this includes whitespaces!) and that capital letters convert to capital letters.

Careful: On Themis, while submitting a C++ file, make sure you have .cpp extension. In case of Java, you should have 'SubstitutionCipher' as the class/file name as per Themis settings.

Exercise 3. [1 point]
Purpose of this exercise: understand the properties of the substitution cipher. In the report, provide your answers to the following questions:

1. How many possible alphabets could be used in a substitution cipher that uses shifting? What about the number of possible alphabets in a substitution cipher that uses a mixed alphabet?
2. Does applying a significant number of consecutive simple substitution cipher encryptions/decryptions with a mixed or shifted alphabet make it harder to break the original plaintext? Justify your answer.
3. Can the encryption function of the substitution cipher also be used for decryption? If so, how?

Exercise 4. [2 points]
Purpose of this exercise: learn to encrypt and decrypt a Vigenère cipher text with a given key.
The Vigenère cipher uses a key to encrypt a plaintext. The key is applied repeatedly as a shift that transform a character in the plaintext into a corresponding ciphertext. The fact that the exact shift is dependent on the key that is in use makes it non-trivial to recover the plaintext. The first letter of the plaintext is encrypted using the first letter of the key as a shift, the second letter of the plaintext is encrypted using the second letter of the key, and so on. When we reach the letter at the position that is one ahead of the length of the key, we loop back to the start of the key, so this letter will be encrypted using the first letter of the key. In this exercise, you need to implement both encryption and decryption (but you don't need to concatenate encryptions and decryptions).

As an example, take the following key:

```
lemon
```

... and the plaintext:

```
This, you see, is a plaintext that is used for teaching purposes.
```

Let's see what result we get when we apply *encryption*:

```
Elug, lzy esr, tw m dylmzhrix fvne me ifph rce eimqutrs dhctagrd.
```

The 'T' was shifted by 'l' - 'a' = 11 positions to the right, which gives

```
(20 + 11) % 26 = 5
```

The fifth letter of the alphabet is 'E', so the encryption produces this character. Verify at least a few of the letters in the example for yourself so that you feel confident that you understand how Vigenère encryption works.

Implement the Vigenère encrypter/decrypter and submit your program to Themis. **Make sure both encryption and decryption are supported.** The input consists of multiple lines. On the first line, you place an 'e' or a 'd' for encryption or decryption, respectively. Then you place the key used for encryption or decryption, separated by a whitespace character. The following lines/line, until the end of file (EOF), hold the input text that you want to encrypt or decrypt (that is, they might be separated by a newline character). Keep the following points in mind when writing your program:

- Special characters (numbers, whitespaces, newlines, commas, etc.) should not be changed by your encryption algorithm.
- You should count the current position based on the number of letters, not the number of characters. In the example given above, this means that the 'y' in the word "you" is the fifth letter and would therefore be encrypted using the 'n' in "lemon".

On Themis, while submitting a C++ file, make sure you have .cpp extension. Additionally, for java, you should have 'VigenereEncryption' as the class/file name.

Exercise 5. [3 points]
Purpose of this exercise: learn to break a standard Vigenère cipher text without a key.

In this exercise, you will try to recover the key used for a cipher text encrypted using the Vigenère cipher. The nature of the Vigenère cipher is such that you cannot easily retrieve the plaintext from the ciphertext like you can with simple substitution ciphers. However, if you had the key used to encrypt the text, you could simply decrypt the text using the same algorithm. Therefore, we will try to attack the key.

A number of different texts have been encrypted and made available on Themis. Your goal is to find the key used for each of these texts.

The encryption leaves non-letter characters as is, and maintains lower- and uppercase letters. Your first goal is to estimate the most likely key length. The input in Themis gives you a range between which you should search for the key in each different test. Make sure your program can test different ranges of key lengths. When computing the most likely key size use the following hints:

- For a potential key size `k`, `k` vectors of letter frequencies should be computed (a vector for each key character). Having computed these vectors, compute the standard deviations of the frequencies found in each of the `k` vectors and sum the `k` computed standard deviations. The correct key size will have the largest value.

  For example, a particular encrypted text might have been encrypted with a key of size 8. The table of sums of standard deviations could look like this:

```
Sum of  5 std. devs: 17.2
Sum of  6 std. devs: 24.9
Sum of  7 std. devs: 17.9
Sum of  8 std. devs: 40.9
Sum of  9 std. devs: 20
Sum of 10 std. devs: 27.5
Sum of 11 std. devs: 23.1
Sum of 12 std. devs: 37.7
Sum of 13 std. devs: 24.8
Sum of 14 std. devs: 29
Sum of 15 std. devs: 26.3
```

clearly showing that the spike at key length 8 is the highest. **You need to output a similar table for your test cases to stdout.**

- Estimate the most likely shift for each of the characters from the maximum frequency, which probably represents the character `e`. E.g., if the maximum is at character `k` then the associated keyword character is probably `g`. This is not a mathematical certainty, though. Sometimes there is a *tie* (e.g., the frequency at `k` equals 15, but the frequency at `s` as well, making `o` a good alternative character in the key), or the count may be off by one (e.g., `f('s') == 14`, making `g` the most likely key character, but `o` is a good candidate as well). The test cases have been created such that the most probable key is indeed the key which was used to encrypt the plaintext, so you do not need to worry about this in this exercise.
- You can compute the standard deviation of a vector of frequencies as follows

```
sqrt( sum(x * x) / 26 - sqr( sum(x) / 26 ) )
```

  with `x` a letter frequency, `sum` a function iterating over all `x` values, `sqrt` the square root function, `sqr` the function returning the square of its argument (i.e., `sqr(y) = y * y`). Since there are 26 letters in the alphabet, the fixed denominator 26 was used in the above formula.

Themis expects the following format for input and output:

- On the input, the first line contains the lower bound of the key length search space. The second line contains the upper bound of the key length search space. After the second line, the encrypted text is given, which may contain newlines and any other character (so pay attention to this when parsing the input).
- The output consists of three parts: the table of summed standard deviation frequencies, the key length estimate, and the most probable key. The output should have the following form:

```
The sum of 5 std. devs: 61.97
The sum of 6 std. devs: 87.21
The sum of 7 std. devs: 63.68
The sum of 8 std. devs: 67.50
The sum of 9 std. devs: 132.13
The sum of 10 std. devs: 66.58
The sum of 11 std. devs: 68.01
The sum of 12 std. devs: 92.56
The sum of 13 std. devs: 69.65
The sum of 14 std. devs: 71.31
The sum of 15 std. devs: 94.75
```

```
Key guess:
integrity
```

Themis expects exactly this form of output. The first test on Themis has been made public so that you can check your formatting more easily. Note that the sum of the standard deviations must be printed with a precision of two digits of the decimal (fractional) part.

Additionally, while submitting a C++ file on Themis, make sure you have .cpp extension. Additionally, for java, you should have 'VigenereBreaking' as the class/file name.

Careful: the algorithm that you will be implementing has a problem with periodicity: let's say the actual key size used is 11. If you search in a key length search space that includes 22, then you will find that 22, as a multiple of 11, has a large peak as well, and often larger than 11. How do you know that the actual key length is 11 and not 22? You will not. In the test cases, the search space has been carefully chosen so that you won't run into this problem. If you start to create your own test cases when debugging, however, you must be mindful of this.