Assignment week 5: Text encryption and decryption

Course 'Imperative Programming' (IPC031)

Make sure to start working on the assignment **before the lab** session, otherwise you will be too late for the deadline

1 Background

In this assignment you work with arbitrarily large input and output text files, as well as a theoretically interesting encryption method, the so-called one-time pad (OTP) method. OTP is an encryption method that is unbreakable in the ideal case. The fundamental assumption is that if you have a sequence of pure random-numbers, you can use them to manipulate a message in such a way that it contains data that seems to be completely random. To decode the encrypted message, you need to have an exact copy of the sequence of numbers that were used to encode the message. This copy, together with an inverse of the encrypting operation is all that is required to recover the original message. Typical encrypting operations are bitwise exclusive-or and addition modulo 128 (note that the maximum ASCII-code of a `char` value is 127). However, OTP is not a practical way of encrypting and decrypting messages for several reasons:

- Generating pure random numbers is very hard. Most computer languages offer pseudo-random number generators, but they are not sufficiently irregular.
- To prevent unwanted decrypting, the used sequence of random numbers must be destroyed. It is very hard to destroy data that is carried on computer media.
- The encryptor and decryptor of the message need to use identical copies of the random number sequence. This makes the system vulnerable for damage, theft, and copying.

Nevertheless, OTP has been used in practice by the KGB (the intelligence agency of the Soviet Union). Sequences of pure random numbers were printed on a tiny scale for safety measures.
In "`main.cpp`" you find an implementation of pseudo-random numbers that you must use in this assignment.

- `void initialise_pseudo_random (int r)`: this function must be called once before any encryption / decryption operation. Choose a value for `r`, such that $0 < r < 65536$. It sets the function `next_pseudo_random_number` to produce the same sequence of pseudo random numbers in each run of your program.
- `int next_pseudo_random_number ()`: a call of this function returns the next pseudo random number. Do not call this function before the initialization function.

2 Learning objectives

After doing this assignment you are able to:

- Work with text files for input and output purposes. These are `ifstream` and `ofstream` objects;
- Work with ASCII characters (`char`) and their conversion to integers (`int`) and vice versa.
- Write unit tests using the Googletest framework.

On Brightspace you can find documentation on unit testing, including a .zip file with a test assignment to test your setup. More on the test assignment is explained in the documentation on Brightspace.

3 Assignment

Part 1: Encryption and decryption of a single character

Encrypting and decrypting a single ASCII character a with a single random number r (with $r \geq 0$) gives the result code b:

$$b = \begin{cases} a & \text{if } a < 32 \\ (a - 32 \otimes (r \,\%\, (128 - 32)) + (128 - 32)) \,\%\, (128 - 32) + 32 & \text{if } a \geq 32 \end{cases}$$

The case distinction ensures that the code of so-called control ASCII characters ($0 \leq a < 32$) remain unchanged, which is necessary for the generation of a legible text output file. Other ASCII characters

$(32 \leq a < 128)$ are encrypted / decrypted according to the formula above that we call the function `rotate_char`. In the case of encryption, the operation $\otimes$ is addition. In the case of decryption, the operation $\otimes$ is subtraction. The extra operation + (128-32) is included in order to prevent the occurrence of negative values, in the case of decryption.
Implement this function by `char rotate_char (char a, int r, Action e)`. Make use of the enumeration: `enum Action {Encrypt, Decrypt}`, enabling `rotate_char` to distinguish between encrypt and decrypt.
Decryption is the inverse operation of encryption for any ASCII character `a` and random number `r` ≥ 0. In words:

```
a = rotate_char (rotate_char (a, r, Encrypt), r, Decrypt)
```

Testing. The file "`main_test.cpp`" contains some tests for the function `rotate_char`. Run the tests from VS Code and verify that your implementation passes these tests. Add your own input/output and propery-based tests to the file, the requirements for these tests are described in "`main_test.cpp`". As usual, make sure that you obtain good test coverage.

Part 2: opening input and output files

Design and implement a function:

```
bool open_input_and_output_file (ifstream& infile, ofstream& outfile)
```

The function asks the user to enter two file names for input and output. The file names may not contain layout symbols. The function checks if the input and output file names are different. Generate an error message if the file names are identical. If the input file and output file have been opened successfully, the result of the function is `true`, otherwise it is `false`. The function provides console information whether the files have been opened successfully or not.

Part 3: encrypting and decrypting with OTP

The "`main.cpp`" file contains part of the program to call the OTP encryption / decryption algorithm, using the functions developed in Part 1 and Part 2. It performs the following steps:

- The user enters a choice to encrypt or decrypt an input text file, using the given function `get_user_action`.
- The user enters file names for the input text file A and output text file B, using `open_input_and_output_file`.
- The user enters an initial value for R, using `initial_encryption_value`.
- The pseudo-random stream is initialized with R, using `initialise_pseudo_random`.
- Read the stream of ASCII characters from input file A and write their encrypted stream to output file B. Make use of `rotate_char`. During encryption / decryption, obtain subsequent random numbers through `next_pseudo_random_number`. This function is called `use_OTP`.
- Close the files A and B.

On Brightspace you find "`assignment-05-mandatory-files.zip`", which contains a number of files:

- The file "`main.cpp`" contains functions for working with pseudo-random numbers and the functions described above.
- The file "`test.txt`" contains a small poem taken from Douglas Adams' novel "The Hitchhikers Guide To The Galaxy". The files named "`test_encrypted_with_R.txt`", for different values of R, are the results of encrypting "`test.txt`" with initializing value R. For instance, "`test_encrypted_with_1.txt`" is the file obtained by encrypting the input file "`test.txt`" using initializing value R = 1. Use these files for testing your OTP algorithm.

Suggestions:

- Two text files can be compared for (in)equality with the Unix command `diff` or Windows command `fc`. For instance:

  ```
  fc test.txt test_encrypted_with_1.txt
  ```

  compares the two text files "`test.txt`" and "`test_encrypted_with_1.txt`".

- **For non-Windows users**: Windows handles 'newlines' differently from Unix based systems, as it uses two control ASCII characters for this purpose ('carriage return' `'\r'` and 'new line'[1] `'\n'`, indicated as CRLF or dos line endings) instead of one control ASCII 'new line' `'\n'` character, indicated as LF or unix line endings. During this course we will provide and grade you with LF line endings text files, but if you want to operate on files generated on a Windows system, you need to convert the CRLF files to LF line endings. You can use the command `dos2unix` or some other tool/editor. For instance:

```
dos2unix test.txt
```

Testing. The file "`main_test.cpp`" contains some tests for the function `use_OTP`. Run the tests from VS Code and verify that your implementation passes these tests.

4 Products

- "`main.cpp`" that you have created with solutions for each part of the assignment.
- "`main_test.cpp`" that has been extended with non-trivial unit tests for each new function that you have developed in "`main.cpp`".
- **If your implementation fails a provided test you cannot obtain grade 'Good'.**

Deadline

Lab assignment: Friday, October 4, 2024, 23:59h

Important notes:

1. check that you have actually submitted your solution in Brightspace.
2. the deadline is firm, and it is impossible to upload a solution after expiry. In that case you fail the assignment.
3. you can upload solutions to Brightspace several times, but only the last submission is stored.
4. identify yourself and your lab partner in every uploaded document. The identification consists of your first and last names, student numbers, and number of (sub) assignment. By identifying yourself, you declare that the uploaded work has been created solely by you and your lab partner.
5. your work will be judged and commented upon. We expect that you obtain the feedback, read it, and use it to for the next exercises.
6. it is essential that you only submit your own solution, never copy somebody/something else's solution, and never share your solution—in particular: **AI tools (including but not limited to Github Copilot or ChatGPT) are not permitted**, solutions from previous year cannot be reused, and finally, you and your lab partner take joint responsibility for the assignment you submit.

[1] Sometimes called 'line feed'.