Exercise 3: Dual-Grid Polybius Square Cipher

Related Topics

I/O, String Manipulation, Program Arguments

Background

The Polybius Square is a simple cipher method that uses a grid to encode letters into numbers. Traditionally, a single 5 $\times$ 5 grid is used, containing the alphabet omitting the letter 'J', which is usually fit into the grid of 'I'. The following figure shows the grid used in history:

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I,J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

In this cipher, each letter is represented by its coordinates in the grid. For example, 'A' in the top-left corner is encoded as "11", 'H' is "23", and 'Z' is "55", etc. However, the traditional cipher can only represent 25 uppercase letters and the encrypted message cannot necessarily be decrypted back to the original message, as "24" can represent both 'I' and 'J'.

To address these problems, a **dual-grid cipher** is introduced. In this cipher, to accommodate all 52 uppercase and lowercase English letters, we implement two distinct 5x5 grids:

- The **Uppercase Grid** encodes uppercase letters from 'A' to 'Z' omitting 'J'.
- The **Lowercase Grid** follows a similar pattern as the uppercase grid.

Encoding and Decoding Rules

In this exercise, we guarantee that the input message contains only English letters and spaces. The **encoding process** follows these rules:

1. **Space**: The space character is encoded as "000".
2. **Uppercase 'J'**: The uppercase letter 'J' is encoded as "100".
3. **Lowercase 'J'**: The lowercase letter 'j' is encoded as "200".
4. **Other uppercase letters**: Located in the uppercase grid, the letter is encoded as its coordinates in the grid prefixed with "1". For example, 'A' is encoded as "111" and 'H' is encoded as "123".
5. **Other lowercase letters**: Located in the lowercase grid, the letter is encoded as its coordinates in the grid prefixed with "2". For example, 'a' is encoded as "211" and 'h' is encoded as "223".

For example, the message "Hello World" is encoded as:

```
123215231231234000152234242231214
```

The **decoding process** reverses the encoding process. The encoded message is split into 3-digit blocks. The prefix digit of each block indicates the grid, and the next two digits indicate the coordinates in the grid. Note that space and uppercase 'J' and lowercase 'j' can be directly translated back.

For example, the code `123115131131134000152134142131114` is decoded as:

```
HELLO WORLD
```

Task Overview

In this exercise, you are asked to implement a **CLI** (Command Line Interface) tool, which encrypts and decrypts messages using the dual-grid Polybius square cipher. The tool involves the following functionalities:

- **Input preprocessing**: The tool should read the message or the encoded message from the standard input (stdin). In the program, you can use the `std::cin` or `std::getline` to read the input message. Before processing the message, **the tool**

should remove any leading or trailing spaces from the input message. If the message consists of only spaces, the tool should print an empty line and exit.

- **Encryption**: Given a message, the tool encrypts the message using the dual-grid Polybius square cipher and prints the encoded message. The user should use the `-e` or `--encrypt` option to enable the encryption mode. Sample usage:

```
./ex3 -e
# Input a message by stdin
Hello World
123215231231234000152234242231214
```

- **Decryption**: Given an encoded message, the tool decrypts the message using the dual-grid Polybius square cipher and prints the original message. The user should use the `-d` or `--decrypt` option to enable the decryption mode. Sample usage:

```
./ex3 -d
# Input an encoded message by stdin
123115131131134000152134142131114
HELLO WORLD
```

- **Help message**: The tool should provide a help message when the user uses the `-h` or `--help` option. Sample usage:

```
./ex3 -h
Encrypt or decrypt a message using the Polybius Square cipher.
Usage: ./ex3 <command> [outputMode]
Commands:
        -e, --encrypt        Encrypt a message
        -d, --decrypt        Decrypt a message
Output modes:
        -c, --compact        Process the message without spaces
        -s, --sparse        Process the message with spaces
```

- **Output modes**: The tool should provide two output modes: **compact** and **sparse**. The **compact** mode processes the message without spaces, and the **sparse** mode processes the message with spaces. The user may use the `-c` or `--compact` option to enable the compact mode, and the `-s` or `--sparse` option to enable the

sparse mode. If the user does not specify the output mode, **the tool should use the compact mode by default**. Sample usage:

```
./ex3 -e -s
# Input a message by stdin
Hello World
123 215 231 231 234 000 152 234 242 231 214
```

- Note that when decrypting a sparse message, the user should specify the `-s` or `--sparse` option to enable the sparse mode. Here the default mode is compact mode as well. Sample usage:

```
./ex3 -d -s
# Input an encoded message by stdin
123 115 131 131 134 000 152 134 142 131 114
HELLO WORLD
```

Option Summary

The options for the tool are summarized as follows:

- **Commands**:
  - `-e`, `--encrypt`: Encrypt a message
  - `-d`, `--decrypt`: Decrypt a message
  - `-h`, `--help`: Show the help message and exit
- **Output Modes**:
  - `-c`, `--compact`: Process the message without spaces
  - `-s`, `--sparse`: Process the message with spaces

Error Handling

To make your life easier, we **guarantee that all the input messages or codes are valid and can be encrypted or decrypted correctly**. You don't need to handle any invalid messages in this exercise. **But**, you should still handle the command line arguments properly as follows:

- If the user provides no arguments, the tool should print `No option!` and exit.

```
./ex3
No option!
```

- If the user provides an invalid option, the tool should print `Invalid option!` and exit.

```
./ex3 -a
Invalid option!
./ex3 -e -l
Invalid option!
```

- If the user provides more than two arguments, the tool should print `Too many options!` and exit. Specifically, for the `-h` or `--help` option, the number of arguments should be exactly 1.

```
./ex3 -e -s -c
Too many options!
```

- If you encounter multiple command line option errors, you should print the error message in the priority order of `Too many options!` > `Invalid option!`.

```
./ex3 -e -1 -2
Too many options!
```

Implementation Details

- The grids and the print help function are provided in `cipher.h` and `cipher.cpp`. You should implement the main function in `ex3.cpp` and submit them together.
- You can modify the provided files and add helper functions as you like.
- In your program, do not use `std::cerr`, `exit(1)` or `return 1;` to handle command line argument errors. You should use `std::cout` to print the error message and `return 0;` to exit the program. This is because JOJ cannot recognize the error message printed by `cerr` and exit code `1`. **The cases which failed due to this reason will not be considered as correct**.
- Note that after printing the encoded or decoded message, you should print a newline.
- When decrypting a message, the user should specify the output mode corresponding to the input message's mode.

- The command line arguments' order is **fixed**. The first command line option must always be a command, which can be encrypt/decrypt/help. If there is a second argument, it must specify the output mode, which can be compact/sparse. **THE ORDER DO NOT CHANGE.**

Submission

Compress your `cipher.h`, `cipher.cpp`, and `ex3.cpp` files into a zip file and submit it to JOJ. **The due date is Nov. 3rd, 23:59.**