UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

CIS4XXX

Exercise: XOR Cipher App

Overview

In this exercise, students will practice working with the **curses** TUI, file loading, C/C++ binding, and benchmarking by building an application to apply and benchmark a simple XOR cipher in Python and C/C++. Note that you will need to use the **WSL terminal** (not PyCharm) to run and test this application!

Requirements

Each student will develop an application using the **curses** module. This application will provide a text-based interactive interface for users to load text, apply a cipher variant, and benchmark the variants against one another. The driver program and Python routines will be in the `cipher.py` file and the C/C++ cipher in `cipher.cpp`.

Driver Program

When run, the program should…

1) Display welcome message and menu
2) Initialize default text and key values (Fig. 1)
3) Show current text and key values under menu
4) Show status message (left-aligned) on last line
5) Detect **single-character input** for menu
6) Accept upper and lowercase input variants
7) Use a 25x80 (row/column) layout (Fig. 1)
8) Have ***no global variables***
9) Only run the application if invoked directly

Fig. 1: Application upon initial execution (exactly as shown); Menu box: 10x40 (r,c)

Reading Strings from the User

Whenever a string is read from the user, the same dialog structure should be used (Figure 2).

The dialog should be formatted as follows:

1) Start on 18th row, 2nd column and be 6 x 78
2) Display prompt, centered, on 2nd row
3) Display box, centered, under prompt, size 3x68
4) Allow entry of 65 characters on one line only
5) Trim resulting whitespace at beginning / end
6) Terminate on entry of the ENTER / linefeed
7) All texted should be manipulated as ***CP437***
8) The prompt must match the specification
9) The status should be updated upon completion
10) If an empty string is entered, this should be considered a cancellation action by the user.

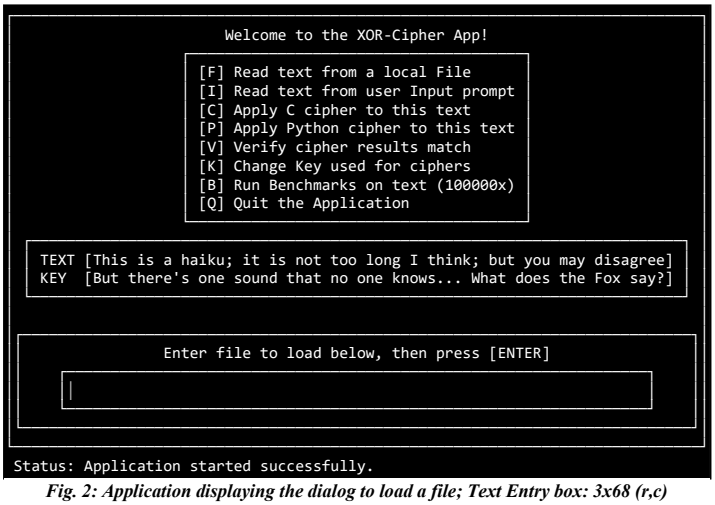

Fig. 2: Application displaying the dialog to load a file; Text Entry box: 3x68 (r,c)

NOTE: The curses textbox adds an extra space at the end of the input – make sure to test thoroughly!

File Handling

Files should be read in text mode with assumed encoding of CP437, allowing for the assumption of 8-bit characters with a range of 0-255. The dialog prompt and status updates upon completion should be as follows:

Prompt	**"Enter file to load below, then press [ENTER]"**
Status, Cancellation	**"Status: File load cancelled."**
Status, Success	**"Status: File contents loaded successfully."**
Status, Failure	**"Status: ERROR: COULD NOT LOAD FILE: filename.txt."**

Text and Key Input

The text and key input dialogs for direct entry should be as follows:

Prompt, Text	**"Enter new text below, then press [ENTER]"**
Prompt, Key	**"Enter new key and then press [ENTER]"**
Status, Text, Cancellation	**"Status: Cancelled user input of text (empty string)."**
Status, Text, Success	**"Status: New text loaded into memory from user input."**
Status, Key, Cancellation	**"Status: Cancelled user input of key (empty string)."**
Status, Key, Success	**"Status: New key loaded into memory from user input."**

Cipher Application

The ciphers, in Python and C, will apply a variant of the XOR-Cipher on a ***byte sequence*** that can use a message and key of any non-zero length. The cipher is straightforward, and a Python variant is provided here:

```
def cipher(message, key):
    return bytes([message[i] ^ key[i % len(key)] for i in range(0, len(message))])
```

While the cipher is simple, some characters cannot be displayed correctly without adjustment – particularly, the first 32 "control characters". Symbols were developed to represent these characters graphically. To remove control characters and replace them with the symbolic counterparts, a translation can be used, as follows:

```
ctrl_translation = str.maketrans(bytes(range(0,32)).decode("cp437"), "␀☺☻♥♦♣♠•◘○◙♂♀♪♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼")
display_text = text.translate(ctrl_translation)
```

This translation should be used whenever displaying text in order to avoid problematic characters!

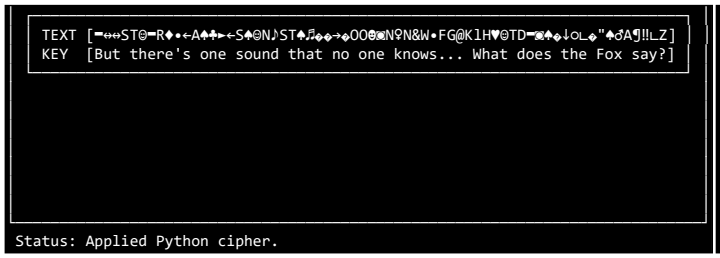

Fig. 3: After application of Python cipher; Text/Key Display box: 4x76 (r,c)

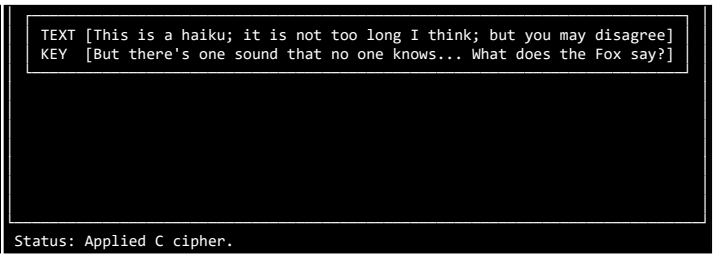

Fig. 4: After application of C/C++ cipher.

Cipher Verification

For verification, the current text and key should be run through the ciphers – ***without updating the text*** – and the results should be compared to verify that the cipher-text from each match one another.

Status, Success	**"Status: Cipher match verified!"**
Status, Failure	**"Status: WARNING: ciphers do not match!"**

Note that while your C library may work, testing with another library (one that fails) is fair game!

Cipher Benchmark

To benchmark the ciphers, each should be run 100,000 times using the **timeit** module, and the results should be displayed. The dialog should appear immediately, just before the benchmark begins, displaying a note that the benchmark is running (see below). Once the benchmark is complete, the dialog should be updated with the data. *This benchmark data should disappear from the screen as soon as another action is taken*.

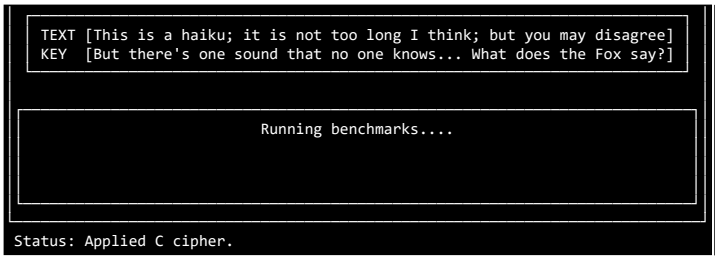

Fig. 5: Dialog when benchmarking is in progress; Response box: 6x78 (r,c)

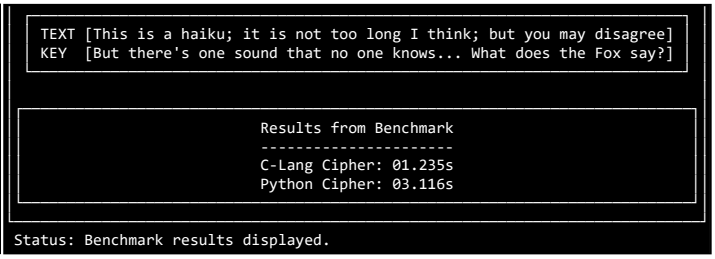

Fig. 6: Dialog when benchmarking has completed

Invalid Menu Selection

If an invalid character is pressed at the menu, the status should be updated to:
"Status: ERROR: Invalid menu selection!"

Termination

When the user elects to quit, the windowing system should shut down. On the ordinary text console, a goodbye message should be printed: **"Thanks for using the XOR-Cipher App; See you next time!"**

If an invalid character is pressed at the menu, the status should be updated to:
"Status: ERROR: Invalid menu selection!"

Structure

The following functions *must be present* in the program.

Python (cipher.py)

run_gui(background)
Takes the background (sometimes called **stdscr**) object as a parameter and runs the program. It should be runnable using the **curses.wrapper()** function. This function should not return anything.

cipher(message, key)
Executes the Python cipher described earlier. It should accept two **byte sequences** and return a ciphered **byte sequence**. It should not change the data stored in message or key.

load_cipher_lib(library_path)
Loads the cipher shared library at library_path, set its method parameters, and return the **library object**.

C++ (cipher.cpp)

void **cipher**(const char *msg, const char *key, char *buffer, int msg_len, int key_len)
Executes the C-based cipher. The cipher-text should be written to buffer by applying XOR to key and msg.

Submissions

NOTE: Your output must match the example output *exactly*. If it does not, ***you will not receive full credit for your submission***! Please submit only and exactly these files:

Files:	cipher.py, cipher.cpp
Method:	Submit on Canvas