CMPUT 201 - Winter 2023
Assignment 1
Henry Tang -- University of Alberta

Concepts Covered

- Arrays
- Input/output
- Program arguments
- Expressions
- Functions
- Error handling
- Selection statements and loops

GitHub Instructions

- By now, you should have already associated your GitHub account with your CCID when you used the invitation link for the Labs repo.
- The github classroom assignment invitation link is https://classroom.github.com/a/EHfagNKw. **READ BELOW BEFORE CLICKING ON THIS LINK**
- Once you click on the link, you will have a repository with the following name [https://gihub.com/cmput201-w23/assignment1-<your github username>] (https://gihub.com/cmput201-w23/assignment1-<your github username>). **Please read the `README.md` to understand the files provided in the repo and how to find and interpret the Github Actions build status.**
- Please update the `LICENSE.md` file with your name where it says `<student name>`
- Make sure to continuously commit and push to your repository as you are developing your assignment and not simply when you are done. This serves two purposes: (1) you always have a backup of your code and can navigate through history to see your changes or revert back to an old version of the code that worked better and (2) showing your progress through the assignment and having the ability to explain the changes you were doing is of great help to you if you are suspected of plagiarism. To avoid overloading GitHub actions, you can commit locally as much as you want but push only once every hour.

Problem Description

Bob and Alice want to send messages to each other, but want to make sure that no one else can understand their messages. They decide on a secret scheme to exchange any data they send to each other (i.e., a *cipher*), as follows:

- They will create a *mapping scheme* that maps each letter in the plaintext alphabet to a different letter in the ciphertext alphabet. Letters can appear in any order in the mapping scheme but there should be a total of 26 **distinct** letters in both alphabets. For example, the following mapping, `mapping1`, is a valid

mapping scheme, where the plaintext alphabet appears before the comma and the ciphertext alphabet appears after the comma.

```
a,p
b,h
c,q
d,g
...
```

and the following mapping, `mapping2`, where plaintext alphabet does not appear in order, is also valid:

```
d,f
b,s
x,u
e,i
m,o
t,y
o,p
n,h
...
```

- The following are the steps taken to encrypt a plaintext word:
    - For each letter in the word, replace that letter with the corresponding letter from the ciphertext alphabet according to the input mapping scheme.
    - Once all the letters in the word are encrypted, output the result in *reverse order*.
    - For example, given `mapping2`, the input word `edmonton` will be encrypted into the *cipher text* `hpyhpofi`
- The following are the steps taken to decrypt a given ciphertext word:
    - For each letter in the ciphertext word, replace that letter with the corresponding letter from the plaintext alphabet according to the input mapping scheme.
    - Once all the letters in the word are decrypted, output the result in *reverse order*.

Task

Write a program that implements the above encryption and decryption scheme.

How to run your program

Your program runs as follows:

`./encrypt -t <mappingfile> -m <encryption mode> -i <inputfile>`

where

- *mappingfile* is a comma separated file (csv) as shown above and is preceded by the flag `-t`.
- *encryption mode* determines if the program should encrypt or decrypt the input provided by the user. The mode is `1` for encryption and `2` for decryption. The mode is preceded by the flag `-m`.
- *inputfile* is a text input file that contains the list of words that are going to be encrypted or decrypted. Each word appears on a separate line. The file is preceded by the flag `-i`.
- All the above arguments are required for the program to run and must be preceded by their respective flags.
- Note that the order of the program arguments should not matter. For example, the following are all equivalent ways to run the program and will yield the same behavior:

```
./encrypt -t mappingfile.csv -m 1 -i words.txt

./encrypt -m 2 -i words.txt -t mappingfile.csv

./encrypt -m 1 -i words_to_encrypt.txt -t mappingfile.csv
```

Error Checking

Your program must report and print an error to `stderr` and quit in the following cases. Note the return code that needs to be returned in each case. The return code is the integer that your program returns upon exit (e.g., `return 0;` in main or `exit(2);` anywhere in the program). The error message must be the exact error message provided (the quotations just show you the boundary of the message and the angle brackets just indicate placeholders and must be replaced by the exact argument. Both the quotations and the angle brackets should not be included in the message):

- If the input file `mappingfile` is not found:
    - Error Message: **"Error: Mapping file <Name of mapping file> does not exist"**
    - Return code: 3
- If the format of the input file `mappingfile` is incorrect. You need to check that (1) the mapping file has exactly 26 lines, (2) each line has the format `<plaintext lower case letter>,<ciphertext lower case letter>` (characters other than lower case letters should not be accepted), (3) there are no repeated letters in either alphabets, and (4) all 26 letters of the English alphabet appear in both the plaintext and ciphertext alphabets.
    - Error Message: **"Error: The format of mapping file <Name of mapping file> is incorrect"**
    - Return code: 4
- If the input file `inputfile` is not found:
    - Error Message: **"Error: Input word file <Name of input file> does not exist"**
    - Return code: 5
- If the user passes in an incorrect mode (i.e., any mode other than 1 or 2):
    - Error Message: **"You entered <value of incorrect mode>. Sorry, your mode must be 1 for encryption or 2 for decryption"**
    - Return code: 6

- For any other incorrect invocations of the program (e.g., invalid arguments, extra arguments, or not providing the required arguments):
    - Error Message: **"Usage: ./encrypt -t <mappingfile> -m <encryption mode> -i <inputfile>"**
    - Return code: 7

Expected Output

- Depending on the mode entered, the following will happen:
    - For mode 1:
        - The program encrypts each input word in the given `inputfile`, in the same order they appear, and outputs the result on the screen. Each line in the input file corresponds to a line in the output. **Make sure to print to `stdout` and not to include any spaces after the output word on each line.**
        - Once the program finishes processing the input list of words, it will exit. Note that your program must return 0 on successful completion.
    - For mode 2:
        - The program will decrypt each input ciphertext word in the given `inputfile` and output the result on the screen. Each line in the input file corresponds to a line in the output. **Make sure to print to `stdout` and not to include any spaces after the output word on each line.**
        - Once the program finishes processing the input list of words, it will exit. Note that your program must return 0 on successful completion.

Please do not include any extra messages or debugging information. Any extra output will make your output different from the expected output, and will cause you to fail the test cases.

Program Format

- Your program consists of a single C file called `encrypt.c`
- Your program must contain at least three functions (the `main` function and two additional functions). You can choose how you will modularaize your program (i.e., which functionality you will include in these two functions). Your functions must make sense and represent a coherent functionality (i.e., if your two functions are just 2 lines each, that's not really proper modularization). Remember to properly declare your functions at the beginning of the file.
- You are welcome (and encouraged) to have more functions to have a cleaner and more modular program, but the above three functions are the minimum that will count towards your grade. To get started early, you can just create the main function that contains all your code. When we cover functions, you can then create two additional functions, and take out the necessary parts of the code you wrote to put into these functions. **Do not wait till we cover functions to start this assignment. It is unlikely that you will finish it on time.**
- While we will not grade you for header files for this assignment, you are welcome (and encouraged) to create a header file for your program.
- Please check the general assignment information on eClass for code style, comments, etc.

Assumptions

The following are assumptions you can safely make. You cannot assume anything that is not explicitly stated here. When in doubt, ask on the forum.

- The input files we will use to test your program will follow the correct format indicated above:
  - Our test input files will have words that contain only lower-case English letters. Each word will be on a separate line in the input file, and the maximum length of any word is 20 letters.
- We will only test with input that can be encrypted/decrypted using the corresponding input mapping file we use.
- You cannot make any assumptions about the input mapping file. You must ensure that it is in the correct format described above.
- You cannot make any assumptions about the program arguments. You must ensure you handle all cases described in the "How to run your program" section above. When testing your program arguments, we will make sure the expected output of the test case is not ambiguous and corresponds to one output code.

Submission

- You must create a Makefile to compile your program. Your Makefile must also contain the clean target. Your program must compile into an executable called `encrypt` **without any warnings or errors**. **If your TA cannot run `./encrypt` after running `make`, you will get a 0 for the assignment.**

- You will create 1 encryption and 1 decryption test case as follows.

  - The encryption test case will have the following files:
    1. `mapping_encryption.csv` which is the mapping file that will be used for your encryption test case
    2. `input_encryption.txt` which contains a single line with the input word to encrypt
    3. `output_encryption.txt` which contains a single line with the expected encryption output of the input word
    4. `test_encryption.sh` which is an executable bash script that contains the right command (with the correct program arguments) to run your program with the above mapping and input files. Running `./test_encryption.sh` must produce the expected output, which is the same as that provided in the `output_encryption.txt` file.
  - The decryption test case will have the following files:
    1. `mapping_decryption.csv` which is the mapping file that will be used for your decryption test case
    2. `input_decryption.txt` which contains a single line with the ciphertext that need to be decrypted
    3. `output_decryption.txt` which contains a single line with the expected decryption output of the input ciphertext
    4. `test_decryption.sh` which is an executable bash script that contains the right command (with the correct program arguments) to run your program with the above mapping and input files. Running `./test_decryption.sh` must produce the expected output, which is the same as that provided in the `output_decryption.txt` file.
  - We have provided sample test cases in your github repository that you can use for guidance. However, you cannot reuse these for your submitted test cases

    - Note that you must adhere to **these exact file names** as your TA will use automated scripts to mark your test case. Note that you **cannot** submit the provided samples for your test cases. We will use the `diff` command to ensure that your provided files are different.

- Complete the provided `README.md` file that includes at least the following sections:

    - the information about any verbal collaboration (name and ccid of the student)
    - the purpose of your program
    - the exact commands we need to run your code
    - the files submitted in your assignment, how they are structured, and what they do
    - references (e.g., any online resources you used in accordance to the course policy)

- **Copying any code off the internet or other students is not allowed.** Please take this seriously as plagiarism cases will be reported to the Faculty of Science.

- All your submitted code files must contain a header that specifies your information, as indicated in the general assignment instructions on eClass. Remember that all your code files must contain the text of the CMPUT 201 License as a header. You MUST NOT include this header for input/output files.

- Make sure all the required files for submission are up-to-date on your GitHub repository by the assignment deadline. Note that any updates after the assignment deadline will not be considered. We will grade only whatever is in the repository at the time of the deadline.

Important Notes

- Your program must compile using `-std=c99 -Wall` without any warnings or errors. If we cannot compile your program or if it compiles with a warning, you will receive a 0 for the assignment.
- Any missing files will result in a 0 for the assignment. If you forget the README.md, Makefile, etc., you will get a 0. The continuous integration enabled with your repository should already show you if you have any missing files. Additionally, we have provided scripts in your repository that you can use to test locally. PLEASE MAKE USE OF THESE SCRIPTS TO CHECK YOUR SUBMISSION!
- Please follow the instructions carefully. Any extra spaces you put after any word will result in a failed diff between the expected output and your solution. Your github repo already has 2 sample test cases and a script to show you how we will compare the program output to the expected output. **PLEASE USE THIS SCRIPT!** It will tell you if your output differs from what we expect. The continuous integration status will also let you know if there is anything wrong.

Grading

Total: 50 marks

- README.md file contains necessary information according to the above instructions and the general assignment instructions posted on eClass **(1 mark)**
- Correct Makefile that compiles code into an executable called `encrypt` **(2 marks)** -- 0 for whole assignment if this step does not work
- `clean` target in Makefile works correctly **(1 mark)**

- Argument checking and error-handling for incorrect values **(20 marks total)** -- see description of arguments and expected values above
- Passing of encryption and decryption test cases **(20 total) -- 2 of these marks are for passing the test cases provided by the student**
- Code contains three functions as described above **(4 marks)**
- Code is properly formatted and contains comments at least before function declarations and at the beginning of C file. **(2 marks)**