This is CS50

CS50's Introduction to Computer Science

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.clubhouse.com/@davidjmalan) (https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan) (https://www.tiktok.com/@davidjmalan) (https://davidjmalan.t.me/) (https://twitter.com/davidjmalan)

Substitution

For this problem, you'll write a program that implements a substitution cipher, per the below.

```
$ ./substitution JTREKYAVOGDXPSNCUIZLFBMWHQ
plaintext:  HELLO
ciphertext: VKXXN
```

Getting Started

Open VS Code (https://code.cs50.io/).

Start by clicking inside your terminal window, then execute `cd` by itself. You should find that its "prompt" resembles the below.

```
$
```

Click inside of that terminal window and then execute

```
wget https://cdn.cs50.net/2022/fall/psets/2/substitution.zip
```

followed by Enter in order to download a ZIP called `substitution.zip` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that matter!

Now execute

```
unzip substitution.zip
```

to create a folder called `substitution`. You no longer need the ZIP file, so you can execute

```
rm substitution.zip
```

and respond with "y" followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd substitution
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
substitution/ $
```

If all was successful, you should execute

```
ls
```

and see a file named `substitution.c`. Executing `code substitution.c` should open the file where you will type your code for this problem set. If not, retrace your steps and see if you can determine where you went wrong!

Background

In a substitution cipher, we "encrypt" (i.e., conceal in a reversible way) a message by replacing every letter with another letter. To do so, we use a *key*: in this case, a mapping of each of the letters of the alphabet to the letter it should correspond to when we encrypt it. To "decrypt" the message, the receiver of the message would need to know the key, so that they can reverse the process: translating the encrypt text (generally called *ciphertext*) back into the original message (generally called *plaintext*).

A key, for example, might be the string `NQXPOMAFTRHLZGECYJIUWSKDVB`. This 26-character key means that `A` (the first letter of the alphabet) should be converted into `N` (the first character

of the key), `B` (the second letter of the alphabet) should be converted into `Q` (the second character of the key), and so forth.

A message like `HELLO`, then, would be encrypted as `FOLLE`, replacing each of the letters according to the mapping determined by the key.

Let's write a program called `substitution` that enables you to encrypt messages using a substitution cipher. At the time the user executes the program, they should decide, by providing a command-line argument, on what the key should be in the secret message they'll provide at runtime.

Here are a few examples of how the program might work. For example, if the user inputs a key of `YTNSHKVEFXRBAUQZCLWDMIPGJO` and a plaintext of `HELLO`:

```
$ ./substitution YTNSHKVEFXRBAUQZCLWDMIPGJO
plaintext:  HELLO
ciphertext: EHBBQ
```

Here's how the program might work if the user provides a key of `VCHPRZGJNTLSKFBDQWAXEUYMOI` and a plaintext of `hello, world`:

```
$ ./substitution VCHPRZGJNTLSKFBDQWAXEUYMOI
plaintext:  hello, world
ciphertext: jrssb, ybwsp
```

Notice that neither the comma nor the space were substituted by the cipher. Only substitute alphabetical characters! Notice, too, that the case of the original message has been preserved. Lowercase letters remain lowercase, and uppercase letters remain uppercase.

Whether the characters in the key itself are uppercase or lowercase doesn't matter. A key of `VCHPRZGJNTLSKFBDQWAXEUYMOI` is functionally identical to a key of `vchprzgjntlskfbdqwaxeuymoi` (as is, for that matter, `VcHpRzGjNtLsKfBdQwAxEuYmOi`).

And what if a user doesn't provide a valid key? The program should explain with an error message:

```
$ ./substitution ABC
Key must contain 26 characters.
```

Or really doesn't cooperate, providing no command-line argument at all? The program should remind the user how to use the program:

```
$ ./substitution
Usage: ./substitution key
```

Or really, really doesn't cooperate, providing too many command-line arguments? The program should also remind the user how to use the program:

```
$ ./substitution 1 2 3
Usage: ./substitution key
```

▶ **Watch a Recording**

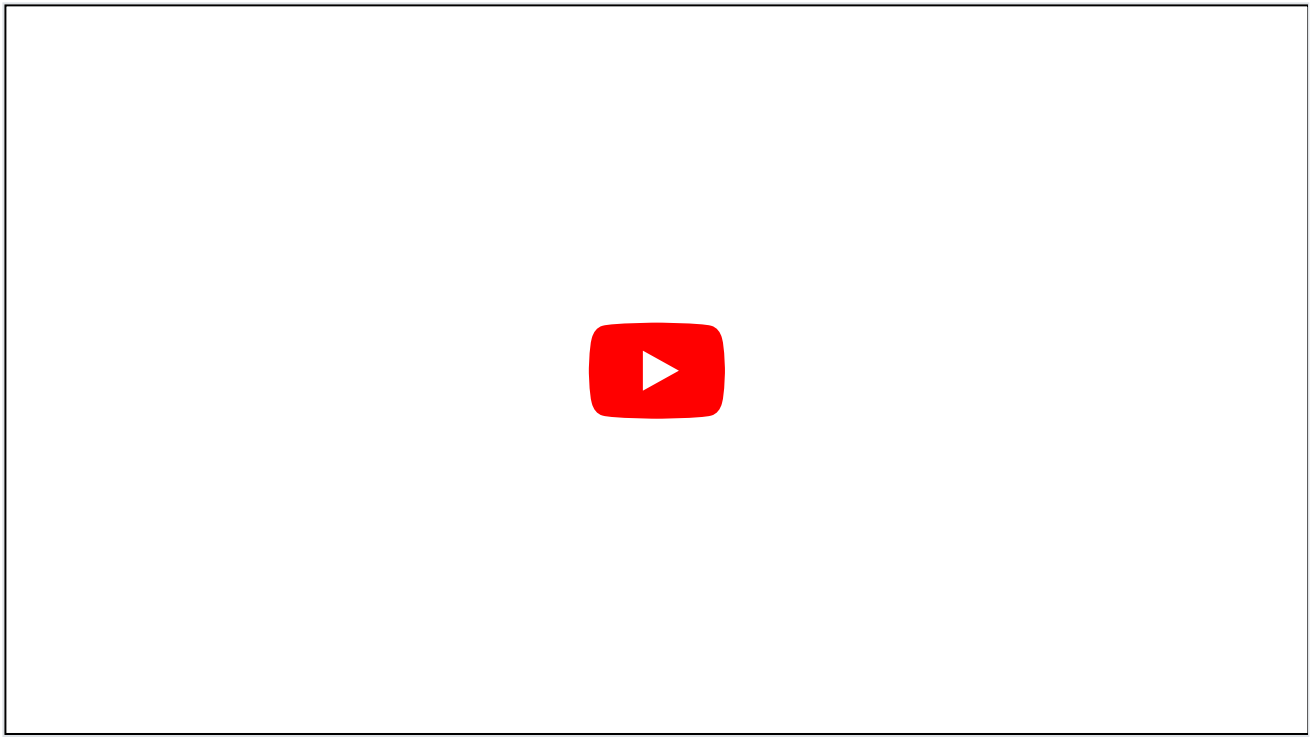
Specification

Design and implement a program, `substitution`, that encrypts messages using a substitution cipher.

- Implement your program in a file called `substitution.c` in a directory called `substitution`.
- Your program must accept a single command-line argument, the key to use for the substitution. The key itself should be case-insensitive, so whether any character in the key is uppercase or lowercase should not affect the behavior of your program.
- If your program is executed without any command-line arguments or with more than one command-line argument, your program should print an error message of your choice (with `printf`) and return from `main` a value of `1` (which tends to signify an error) immediately.
- If the key is invalid (as by not containing 26 characters, containing any character that is not an alphabetic character, or not containing each letter exactly once), your program should print an error message of your choice (with `printf`) and return from `main` a value of `1` immediately.
- Your program must output `plaintext:` (without a newline) and then prompt the user for a `string` of plaintext (using `get_string`).
- Your program must output `ciphertext:` (without a newline) followed by the plaintext's corresponding ciphertext, with each alphabetical character in the plaintext substituted for the corresponding character in the ciphertext; non-alphabetical characters should be outputted unchanged.
- Your program must preserve case: capitalized letters must remain capitalized letters; lowercase letters must remain lowercase letters.
- After outputting ciphertext, you should print a newline. Your program should then exit by returning `0` from `main`.

You might find one or more functions declared in `ctype.h` to be helpful, per manual.cs50.io (https://manual.cs50.io/).

Walkthrough



How to Test Your Code

Execute the below to evaluate the correctness of your code using `check50`. But be sure to compile and test it yourself as well!

```
check50 cs50/problems/2023/x/substitution
```

Execute the below to evaluate the style of your code using `style50`.

```
style50 substitution.c
```

▶ **How to Use `debug50`**

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2023/x/substitution
```