CSC 111 - FALL 2023
FUNDAMENTALS OF PROGRAMMING WITH ENGINEERING APPLICATIONS
ASSIGNMENT 5
UNIVERSITY OF VICTORIA

Due: Sunday, Nov. 19th, 2023 at 11:59pm. **Late assignments will not be accepted.**

This assignment will be submitted electronically through BrightSpace (as described in 'Submission Instructions' below). All code submissions must be your own work. Sharing code with other students (as either sender or receiver) is plagiarism. Use of generative AI tools (like ChatGPT) for any aspect of programming assignments is also prohibited.

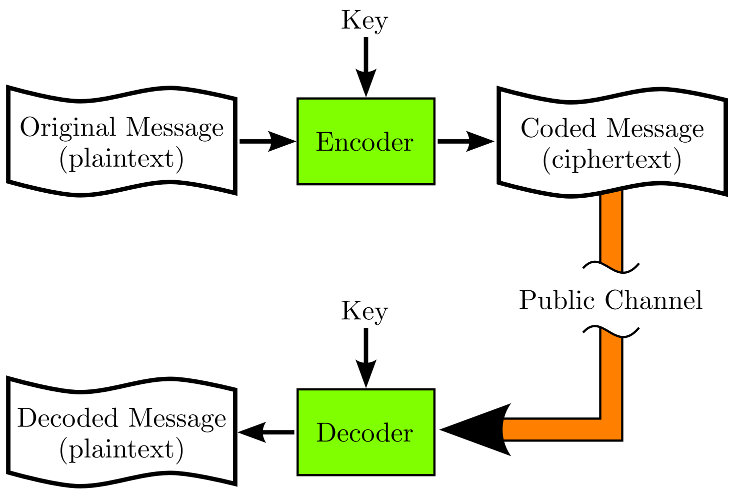
1 Overview

Many communication systems (including the internet) are effectively public channels: In general, any data being transmitted can be intercepted and viewed by a third party. This presents a challenge when sensitive data (like personal information or credit card numbers) must be transmitted. One way to increase the effective privacy of the information is to use cryptography to obfuscate the data when it is transmitted.

A **cryptosystem** is a mechanism to transform a sensitive message (the **plaintext**) into a cryptic representation (the **ciphertext**) such that a third party who intercepts the ciphertext will have great difficulty determining the contents of the plaintext message. To be practical, a cryptosystem must achieve a good balance between the following two factors.

- If an encrypted message is sent from A to B, it must be possible for the intended recipient (B) to decode the message and recover the plaintext.
- If an unauthorized third party Z intercepts the message in transit, it must be difficult for Z to decode the message.

One type of cryptosystem that can achieve such a balance is a **shared key** cryptosystem. In such a system, both the sender and receiver (A and B) share a secret **key** in advance. The key is then used as part of the encryption and decryption process (such that any message encrypted with a particular key can be decrypted by the same key). The conceptual workflow of a shared key system is shown in the diagram below.


Key
Original Message (plaintext)
Encoder
Coded Message (ciphertext)
Public Channel
Key
Decoded Message (plaintext)
Decoder

This assignment covers strings and unformatted file input, and involves implementing the **Vigenere cipher** to encrypt and decrypt strings of text. A description of the Vigenere cipher appears in Section 2 below[1]. Beyond its use in this assignment, you don't need to know anything about cryptography or the Vigenere cipher in this course.

1.1 Your Task

Your task is to write a complete C program that implements the specification described in Section 3 below.

The submission must be contained in a file called `vigenere.c`. Once complete, your program must compile and run in our virtual lab environment (`https://jhub.csc.uvic.ca`). It must be possible to compile your program using the command

```
gcc -Wall -std=c18 -o vigenere vigenere.c
```

and to run your program (after compiling successfully) using the command

```
./vigenere
```

Best Practices for Code Style

Your submissions **must** observe the following guidelines. Submissions which do not meet the guidelines will have marks deducted, even if they function correctly (although some of the guidelines, such as those relating to uninitialized memory, might also affect the ability of the program to work correctly on a reliable basis, which could result in a deduction for both style and function).

- Submitted files must be syntactically correct and named according to the assignment specification.
- Every source file must contain a comment (at or near the top of the file) with your name, student number, the date the code was written and a brief description of the program.
- The `goto` statement is not permitted in any assignment submissions. Instead, your code should use structured control flow elements, like `if`, `for` and `while`.

1. You can also read about Vigenere Ciphers in general on Wikipedia: `https://en.wikipedia.org/wiki/Vigenere_Cipher`.

- Global variables (data variables created outside of the scope of a function) are not permitted, except when explicitly allowed by the assignment. For this assignment, no global variables are permitted, except for `const` variables if needed.
- Every function with a non-`void` return type must return a value.
- Uninitialized variables may not be used before being assigned a value.
- Memory not explicitly allocated by the program (such as the unallocated locations past the end of an allocated array) may not be used in any way, even if the usage does not appear to directly affect the program's behavior.
- Every file opened (by functions such as `fopen`) **must** be explicitly closed (by a call to `fclose`).
- Any dynamically allocated memory must be freed explicitly. For this assignment, it should not be necessary to use any dynamically allocated memory.

2 The Vigenere Cipher

The version of the Vigenere cipher used in this assignment will depend on two string inputs: a **message** and a **key**. Your program will perform both encryption and decryption (using the same two input strings).

For both encryption and decryption, the following indexing scheme will be used to associate a numerical index with each character of the English alphabet.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

For your convenience, code has been provided on BrightSpace for two functions `character_to_index` and `index_to_character` that convert between lowercase character values and their associated indices.

2.0.1 Encryption

This section describes the encryption process with the following input values.

Message:	`computerscience`
Key:	`pear`

Step 1: Expand the key until it is the same length as the message. This is done by repeating the key as needed.

Message:	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e
Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a

Step 2: Compute the numerical index (with respect to the alphabet table above) of each character in the message and each character in the key.

Message:	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e
Message Index:	2	14	12	15	20	19	4	17	18	2	8	4	13	2	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Step 3: Add the two computed indices together (for each character). If the sum is larger than the size of the alphabet, wrap around by subtracting 26 (for example, the index 28 would wrap around to 2).

Message:	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e
Message Index:	2	14	12	15	20	19	4	17	18	2	8	4	13	2	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Index Sum:	17	18	12	32	35	23	4	34	33	6	8	21	28	6	4
Wrapped Sum:	17	18	12	6	9	23	4	8	7	6	8	21	2	6	4

Step 4: To produce the finished ciphertext, convert each wrapped index to its corresponding character in the alphabet.

Message:	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e
Message Index:	2	14	12	15	20	19	4	17	18	2	8	4	13	2	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Index Sum:	17	18	12	32	35	23	4	34	33	6	8	21	28	6	4
Wrapped Sum:	17	18	12	6	9	23	4	8	7	6	8	21	2	6	4
Ciphertext:	r	s	m	g	j	x	e	i	h	g	i	v	c	g	e

The finished ciphertext is the string '`rsmgjxeihgivcge`'.

2.0.2 Decryption

This section describes the decryption process with the following input values. The encoded message is the result of the example in the previous section.

Message:	`rsmgjxeihgivcge`
Key:	`pear`

Step 1-2: As for encryption, expand the key until it matches the size of the message, then compute the index of each character in the key and message.

Message:	r	s	m	g	j	x	e	i	h	g	i	v	c	g	e
Message Index:	17	18	12	6	9	23	4	8	7	6	8	21	2	6	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Step 3: For each message index m and key index k, compute the difference $m - k$. If the difference is less than zero, wrap around by adding 26 (for example, the value -2 would wrap around to $(-2) + 26 = 24$).

Message:	r	s	m	g	j	x	e	i	h	g	i	v	c	g	e
Message Index:	17	18	12	6	9	23	4	8	7	6	8	21	2	6	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Index Difference:	2	14	12	-11	-6	19	4	-9	-8	2	8	4	-13	2	4
Wrapped Sum:	2	14	12	15	20	19	4	17	18	2	8	4	13	2	4

Step 4: To finish decryption, convert each wrapped index to its corresponding character in the alphabet.

Message:	r	s	m	g	j	x	e	i	h	g	i	v	c	g	e
Message Index:	17	18	12	6	9	23	4	8	7	6	8	21	2	6	4

Key:	p	e	a	r	p	e	a	r	p	e	a	r	p	e	a
Key Index:	15	4	0	17	15	4	0	17	15	4	0	17	15	4	0

Index Difference:	2	14	12	-11	-6	19	4	-9	-8	2	8	4	-13	2	4
Wrapped Sum:	2	14	12	15	20	19	4	17	18	2	8	4	13	2	4
Ciphertext:	c	o	m	p	u	t	e	r	s	c	i	e	n	c	e

3 Program Specification

For convenience, we have provided some sample code showing the expected output formatting. You do not have to use this sample code directly, but your program **must** produce output in the **exact** format described below. Even minor deviations (like extra punctuation or capital letters) will result in your output being considered incorrect.

The execution of the program must follow the steps below.

1. When the program starts, it will open an input file called `input.txt`. If it is not possible to open this file, the program will print the error message shown below and **immediately exit** (with no further output).

   Output

   ```
   Error: Unable to open input file.
   ```

2. Once the input file has been opened, the program will read the first two lines of text. The first line will be the **key** and the second line will be the **message**. Both lines are guaranteed to be at most 100 characters in length (so a string declared like '`char key[101];`' would be sufficient to store the key).
3. If the key has length zero (that is, if the first line of the input file is blank), or if the key contains any characters that are not lowercase letters[2], the program will print the error message shown below and **immediately exit** (with no further output).

   Output

   ```
   Error: Invalid key.
   ```

2. Recall that the `islower()` function in `ctype.h` can be used to test whether a character is a lowercase letter.

4. If the message has length zero or contains any characters that are not lowercase letters, the program will print the error message shown below and **immediately exit** (with no further output).

Output

```
Error: Invalid message.
```

5. If both the key and message are valid, the program will compute two result strings: The result of **encrypting** the provided message with the provided key and the result of **decrypting** the provided message using the provided key. The program will then generate exactly four lines of output (directly to the user via `printf`, not to a file). The output must follow the format below (where '`message`', '`key`', '`encryptionresult`' and '`decryptionresult`' are placeholders for the actual data). Notice that the contents of the result strings are surrounded by square brackets.

Output

```
Message: [message]
Key: [key]
Encrypted: [encryptionresult]
Decrypted: [decryptionresult]
```

3.1 Examples

The examples below show the expected behavior of the program on various input files. Notice that if the provided message is an encrypted string (and the provided key matches the key used to encrypt that string), the result of decryption yields the original plaintext.

Example 1: Key '`pear`', Message '`computerscience`'

Input File

```
pear
computerscience
```

Output

```
Message: [computerscience]
Key: [pear]
Encrypted: [rsmgjxeihgivcge]
Decrypted: [nkmyfpeadyinyye]
```

Example 2: Key '`pear`', Message '`rsmgjxeihgivcge`'

Input File

```
pear
rsmgjxeihgivcge
```

Output

```
Message: [rsmgjxeihgivcge]
Key: [pear]
Encrypted: [gwmxybezwkimrke]
Decrypted: [computerscience]
```

Example 3: Key '`raspberry`', Message '`computerscience`'

Input File

```
raspberry
computerscience
```

Output

```
Message: [computerscience]
Key: [raspberry]
Encrypted: [toeevxviqtiwcdi]
Decrypted: [louatpnaulimyba]
```

Example 4: Key ‘`raspberry`’, Message ‘`toeevxviqtiwcdi`’

Input File

```
raspberry
toeevxviqtiwcdi
```

Output

```
Message: [toeevxviqtiwcdi]
Key: [raspberry]
Encrypted: [kowtwbmzokiorem]
Decrypted: [computerscience]
```

Example 5: Key ‘`abc`’, Message ‘`thisinputhasalongmessagebutashortkey`’

Input File

```
abc
thisinputhasalongmessagebutashortkey
```

Output

```
Message: [thisinputhasalongmessagebutashortkey]
Key: [abc]
Encrypted: [tiksjppvvhbuamqnhoetuahgbvvatjosvkfa]
Decrypted: [tggshlptrhzqakmnfkerqafcbtrarfoqrkdw]
```

Example 6: Key ‘`abc`’, Message ‘`tiksjppvvhbuamqnhoetuahgbvvatjosvkfa`’

Input File

```
abc
tiksjppvvhbuamqnhoetuahgbvvatjosvkfa
```

Output

```
Message: [tiksjppvvhbuamqnhoetuahgbvvatjosvkfa]
Key: [abc]
Encrypted: [tjmskrpwxhcwansniqeuwaiibwxaulotxkgc]
Decrypted: [thisinputhasalongmessagebutashortkey]
```

Example 7: Key ‘`thisisareallylongkey`’, Message ‘`shortmessage`’

Input File

```
thisisareallylongkey
shortmessage
```

Output

```
Message: [shortmessage]
Key: [thisisareallylongkey]
Encrypted: [lowjbeejwarp]
Decrypted: [zagzlueboavt]
```

Example 8: Key '`thisisareallylongkey`', Message '`lowjbeejwarp`'

Input File

```
thisisareallylongkey
lowjbeejwarp
```

Output

```
Message: [lowjbeejwarp]
Key: [thisisareallylongkey]
Encrypted: [evebjweaaaca]
Decrypted: [shortmessage]
```

4 Evaluation

Your code must compile and run correctly in the CSC 111 virtual lab environment using the compile and run commands given above. If your code does not compile **as submitted** (with no modifications whatsoever), you will receive a mark of zero. Note that submitting an incorrectly-named file will result in the compile command above failing (and the submission receiving a mark of zero).

This assignment is worth 5% of your final grade and will be marked out of 13.

Second Chance Submission

After the initial due date (Sunday, Nov. 19th, 2023) has passed, your code will be automatically tested using a variety of inputs (which will not necessarily match the example inputs shown in the previous sections). If your programs do not give correct output on some of the tested inputs, you will be allowed to fix your program and resubmit it until Wednesday, Nov. 22nd, 2023. The grade for the assignment will be computed to be the average of the grade for both submissions (so you can recover some, but not all, of your marks by fixing and resubmitting your code). If you do not resubmit your code, your initial submission will be used in place of the resubmission.

If you do not submit an initial version of your code before the due date, you are **not** permitted to resubmit your code and will receive a mark of zero.

Submission Instructions

All submissions for this assignment will be accepted electronically. Submit your `vigenere.c` file through the Assignment 5 entry on BrightSpace. You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions will be accepted after the due date has passed. For the second chance submission described above, a separate assignment entry will be created after the initial due date.

Ensure that each file you submit contains a comment with your name and student number, and that each file is named correctly (as described in this document). If you do not name your files correctly, or if you do not submit them electronically, it will not be possible to mark your submission and you will receive a mark of zero.

To verify that you submitted the correct file, you can download your submission from BrightSpace after submitting and test that it works correctly in the virtual lab environment. It will be assumed that all submissions have been tested this way, and therefore that there is no reasonable chance that an incorrect file was submitted accidentally, so no exceptions will be made to the due date as a result of apparently incorrect versions being submitted.