Vigenere Cipher

Generated by Doxygen 1.13.2

1 Vigenere Cipher

1.1 C How to Program (Ninth Edition) - Global Edition - Deitel,P and Deitel, H

1.1.1 Chapter 8: Characters and Strings Project 8.41 Programming Project: Pqyoaf Nylfomigrob Qwbbfmh Mndogvk: Rboqlrut yua Boklnxhmywex

1.1.2 Programming Tools

- VIM
- GCC (compiled using: gcc -std=c18 -Wall cipher.c cipher_test.c -o cipher_test)

1.1.3 Brief

This project allowed me to learn about creating a secret key called the Vigenere cipher. It is a development on the well-known Caesar cipher.

Using the Caesar cipher each letter is shifted (encrypted) according to:

$f(p) = (p + x) mod 26$

x is the amount of shifting through the alphabet to encode the original text (the plaintext) into a hidden message (the ciphertext). For example if i $x = 1$ then

$a = b$,
$b = c$,
...,
$z = a$
.

Decryption occurs through:

$f^{-1}(p) = (p - k) mod 26$

This may have been good on the battlefield, or as a child sharing the location of your toy cars, but it is fairly (very?!) easy to break.

The Vigenere cipher was created to make this a bit more difficult by creating a two-dimensional matrix of letters

https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

2 File Index

2.1 File List

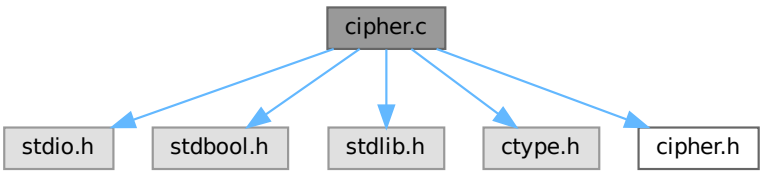
Here is a list of all files with brief descriptions:

3 File Documentation

3.1 cipher.c File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <ctype.h>
#include "cipher.h"
```

Include dependency graph for cipher.c:



Functions

- bool checkKey (char ∗strng)

  Check if string contains only letters of the alphabet.
- char getSubstitution (char scrt_char, char plainOrCipher_char, bool Encrypt)

  Substitute one character with another.
- int encrypt (char ∗plainToEncrypt, char ∗cipherTextArray, char ∗scrt_strng)
- int decrypt (char ∗cipherToDecrypt, char ∗plainTextArray, char ∗scrt_strng)

3.1.1 Function Documentation

checkKey()

```
bool checkKey (
            char * strng)
```

Check if string contains only letters of the alphabet.

Iterate through each character in a string and return false if a character is not a letter from the alphabet.

Parameters

in	*strng*	String to check.

Returns

bool True if only letters, false otherwise.

Definition at line 20 of file cipher.c.

decrypt()

```
int decrypt (
            char * cipherToDecrypt,
            char * plainTextArray,
            char * scrt_strng)
```

Decrypt ciphertext using a secret key

Parameters

in	*cipherToDecrypt*	Ciphertext
in	*plainTextArray*	Empty string of same length
in	*scrt_strng*	Secret key string

Definition at line 141 of file cipher.c.

encrypt()

```
int encrypt (
            char * plainToEncrypt,
            char * cipherTextArray,
            char * scrt_strng)
```

Walks along plaintext string, repeatedly calling getSubstitution.

Parameters

in	*plainToEncrypt*	Plain text string.
in	*cipherTextArray*	Empty string of same length
in	*scrt_strng*	Secret key string

Definition at line 102 of file cipher.c.

getSubstitution()

```
char getSubstitution (
            char scrt_char,
            char plainOrCipher_char,
            bool Encrypt)
```

Substitute one character with another.

This function is used for both encryption and decryption by way of a boolean value. Characters are converted to uppercase. For each character it performs the appropriate substitution.

Parameters

in	*scrt_char*	Secret-Key character.
in	*plainOrCipher_char*	Character from plaintext or ciphertext.
in	*encryptOrDecrypt*	Bool indicating whether to encrypt (true) or decrypt (false).

Definition at line 43 of file cipher.c.

3.2 cipher.c

Go to the documentation of this file.

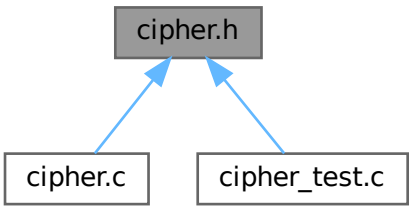
```
00001
00002 #include <stdio.h>
00003 #include <stdbool.h>
00004 #include <stdlib.h>
00005 #include <ctype.h>
00006 #include "cipher.h"
00007
00008 // prototypes
00009 bool checkKey(char *strng);
00010 char getSubstitution (char scrt_char, char plainOrCipher_char, bool de_Or_en);
00011
00012 // Function definitions
00013
00020 bool checkKey(char *strng){
00021     int counter = 0;
00022
00023     while (strng[counter] != '\n'){
00024         if (!isalpha(strng[counter])){
00025             puts("Can't be used");
00026             return false;
00027         }
00028         counter++;
00029     }
00030     return true;
00031
00032 }
00033
00043 char getSubstitution (char scrt_char, char plainOrCipher_char, bool Encrypt){
00044     static const char alphabet[ALPHABET_LENGTH] = {
00045         'A','B','C','D','E',
```

```
00046         'F','G','H','I','J',
00047         'K','L','M','N','O',
00048         'P','Q','R','S','T',
00049         'U','V','W','X','Y','Z'};
00050
00051     static char Vigenere_Square[ALPHABET_LENGTH][ALPHABET_LENGTH] = {{' '}};
00052
00053
00054     if ((Vigenere_Square[0][0]) == ' '){ // call first time only
00055         for (size_t i = 0; i < ALPHABET_LENGTH; i++) {
00056             for (size_t j = 0; j < ALPHABET_LENGTH; j++){
00057                 Vigenere_Square[i][j] = alphabet[(j + i) % ALPHABET_LENGTH];
00058             }
00059         }
00060     }
00061     // convert to integer to traverse array
00062     // the ascii code for each is
00063     // a = 97, A = 65;
00064     int scrt_char_n;
00065     int plain_char_n;
00066
00067     plainOrCipher_char = toupper(plainOrCipher_char);
00068     scrt_char = toupper(scrt_char);
00069
00070     // give ascii numbers
00071     if (isupper(scrt_char)){scrt_char_n = scrt_char - 65;} // A = 0, ... Z = 25
00072     if (isupper(plainOrCipher_char)) {plain_char_n = plainOrCipher_char - 65;}
00073
00074     // encrypt
00075     if((Encrypt)){
00076         return Vigenere_Square[scrt_char_n][plain_char_n];
00077     }
00078
00079     if(!(Encrypt)){
00080
00081 #ifdef DEBUG
00082         puts("Values supplied to function:");
00083         printf("scrt_char: %c\n", scrt_char);
00084         printf("plainOrCipher_char %c\n", plainOrCipher_char);
00085         puts("");
00086         puts("Converted to:");
00087         printf("scrt_char_n: %d\n", scrt_char_n);
00088         printf("plain_char_n: %d\n", plain_char_n);
00089 #endif
00090         for (size_t i = 0; i < ALPHABET_LENGTH; i++){
00091             if(Vigenere_Square[scrt_char_n][i] == plainOrCipher_char){
00092                 return Vigenere_Square[0][i];
00093             }
00094         }
00095     }
00096 }
00097
00102 int encrypt (char *plainToEncrypt, char *cipherTextArray, char *scrt_strng){
00103     int p_counter = 0, s_counter = 0;
00104
00105     if(!checkKey(scrt_strng)){
00106         return EXIT_FAILURE;
00107     }
00108
00109     while (plainToEncrypt[p_counter] != '\0'){
00110
00111         if (!isalpha(plainToEncrypt[p_counter])){
00112             cipherTextArray[p_counter] = plainToEncrypt[p_counter];
00113             p_counter++;
00114             continue;
00115         }
00116
00117         cipherTextArray[p_counter] = getSubstitution(
00118                 scrt_strng[s_counter], plainToEncrypt[p_counter], true);
00119
00120         p_counter++; s_counter++;
00121
00122         if (scrt_strng[s_counter] == '\n'){
00123             s_counter = 0;
00124         }
00125     }
00126
00127     p_counter = 0;
00128
00129     while (cipherTextArray[p_counter] != '\0'){
00130         printf("%c", cipherTextArray[p_counter]);
00131         p_counter++;
00132     }
00133
00134     return EXIT_SUCCESS;
00135 }
00136
```

```
00141 int decrypt (char *cipherToDecrypt, char *plainTextArray, char *scrt_strng){
00142     int p_counter = 0, s_counter = 0;
00143
00144     if(!checkKey(scrt_strng)){
00145         return EXIT_FAILURE;
00146     }
00147
00148     while (cipherToDecrypt[p_counter] != '\0'){
00149
00150         if (!isalpha(cipherToDecrypt[p_counter])){
00151             plainTextArray[p_counter] = cipherToDecrypt[p_counter]; // copy.. then move on
00152             p_counter++;
00153             continue;
00154         }
00155
00156         plainTextArray[p_counter] = getSubstitution (
00157                 scrt_strng[s_counter], cipherToDecrypt[p_counter], false);
00158
00159         p_counter++; s_counter++;
00160
00161         if (scrt_strng[s_counter] == '\n'){
00162             s_counter = 0;
00163         }
00164     }
00165
00166     p_counter = 0;
00167
00168     while (plainTextArray[p_counter] != '\0'){
00169         printf("%c", plainTextArray[p_counter]);
00170         p_counter++;
00171     }
00172
00173     return EXIT_SUCCESS;
00174 }
00175
```

3.3 cipher.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define MAX_SENTENCE_LENGTH 100
- #define MAX_KEY_LENGTH 20
- #define ALPHABET_LENGTH 26

Functions

- int encrypt (char *plainToEncrypt, char *cipherTextArray, char *scrt_strng)
- int decrypt (char *cipherToDecrypt, char *plainTextArray, char *scrt_string)

3.3.1 Macro Definition Documentation

ALPHABET_LENGTH

```
#define ALPHABET_LENGTH 26
```

Definition at line 7 of file cipher.h.

MAX_KEY_LENGTH

```
#define MAX_KEY_LENGTH 20
```

MAX_SENTENCE_LENGTH

```
#define MAX_SENTENCE_LENGTH 100
```

3.3.2 Function Documentation

decrypt()

```
int decrypt (
            char * cipherToDecrypt,
            char * plainTextArray,
            char * scrt_strng)
```

Decrypt ciphertext using a secret key

Parameters

in	*cipherToDecrypt*	Ciphertext
in	*plainTextArray*	Empty string of same length
in	*scrt_strng*	Secret key string

Definition at line 141 of file cipher.c.

encrypt()

```
int encrypt (
            char * plainToEncrypt,
            char * cipherTextArray,
            char * scrt_strng)
```

Walks along plaintext string, repeatedly calling getSubstitution.

Parameters

in	*plainToEncrypt*	Plain text string.
in	*cipherTextArray*	Empty string of same length
in	*scrt_strng*	Secret key string

Definition at line 102 of file cipher.c.

3.4 cipher.h

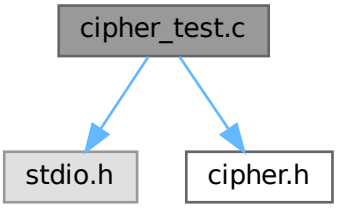
Go to the documentation of this file.

```
00001
00002 #ifndef CIPHER_H
00003 #define CIPHER_H
00004
00005 #define MAX_SENTENCE_LENGTH 100
00006 #define MAX_KEY_LENGTH 20
00007 #define ALPHABET_LENGTH 26
00008
00009 int encrypt (char *plainToEncrypt, char *cipherTextArray, char *scrt_strng);
00010 int decrypt (char *cipherToDecrypt, char *plainTextArray, char *scrt_string);
00011
00012 #endif
```

3.5 cipher_test.c File Reference

```
#include <stdio.h>
#include "cipher.h"
```

Include dependency graph for cipher_test.c:



Macros

- #define DECRYPT

Functions

- int main (void)

3.5.1 Macro Definition Documentation

DECRYPT

```
#define DECRYPT
```

Definition at line 6 of file cipher_test.c.

3.5.2 Function Documentation

main()

```
int main (
            void )
```

Definition at line 10 of file cipher_test.c.

3.6 cipher_test.c

Go to the documentation of this file.

```
00001
00002 #include <stdio.h>
00003 #include "cipher.h"
00004
00005 //#define ENCRYPT
00006 #define DECRYPT
00007
00008 //#define DEBUG
00009
00010 int main(void){
00011     char plainText[MAX_SENTENCE_LENGTH];
00012     char secretKey[MAX_KEY_LENGTH];
00013     char cipherText[MAX_SENTENCE_LENGTH];
00014
00015     puts("Vigenere Cipher Project");
00016     puts("");
00017
00018 #ifdef ENCRYPT
00019     puts("Enter a sentence to encrypt: ");
00020     fgets(plainText, MAX_SENTENCE_LENGTH, stdin);
00021     puts("Enter a secret key of ONLY LETTERS (max 20): ");
00022     fgets(secretKey, MAX_KEY_LENGTH, stdin);
00023
00024     encrypt(plainText, cipherText, secretKey);  // encrypt
00025 #endif
00026
00027 #ifdef DECRYPT
00028     puts("Enter a sentence to decrypt: ");
00029     fgets(cipherText, MAX_SENTENCE_LENGTH, stdin);
00030     puts("Enter a secret key of ONLY LETTERS (max 20): ");
00031     fgets(secretKey, MAX_KEY_LENGTH, stdin);
00032
00033     decrypt(cipherText, plainText, secretKey); // decrypt
00034
00035 #endif
00036
00037     return 0;
00038 }
00039
```

3.7 Mainpage.dox File Reference

Index