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:

$f^{-1}(p) = (p - k)mod26$

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

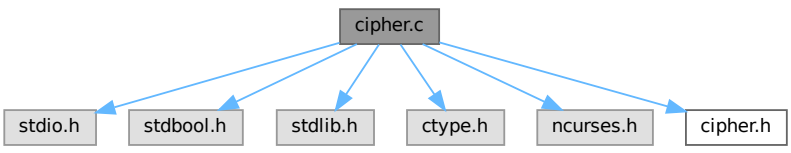
Here is a list of all files with brief descriptions:

3 File Documentation

3.1 cipher.c File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <ctype.h>
#include <ncurses.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 18 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 133 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 98 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 40 of file cipher.c.

3.2 cipher.c

Go to the documentation of this file.

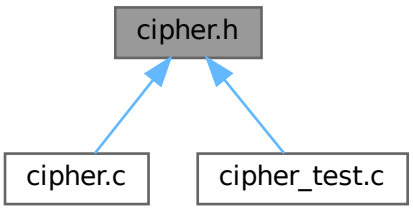
```
00001
00002 #include <stdio.h>
00003 #include <stdbool.h>
00004 #include <stdlib.h>
00005 #include <ctype.h>
00006 #include <ncurses.h>
00007 #include "cipher.h"
00008
00009 bool checkKey(char *strng);
00010 char getSubstitution (char scrt_char, char plainOrCipher_char, bool de_Or_en);
00011
00018 bool checkKey(char *strng){
00019     int counter = 0;
00020
00021     while (strng[counter] != '\0'){
00022         if (!isalpha(strng[counter])){
00023             printw("Can't be used because it contains %d\n", strng[counter]);
00024             return false;
00025         }
00026         counter++;
00027     }
00028     return true;
00029 }
00030
00040 char getSubstitution (char scrt_char, char plainOrCipher_char, bool Encrypt){
00041     static const char alphabet[ALPHABET_LENGTH] = {
00042         'A','B','C','D','E',
00043         'F','G','H','I','J',
00044         'K','L','M','N','O',
00045         'P','Q','R','S','T',
```

```
00046         'U','V','W','X','Y','Z'};
00047
00048     static char Vigenere_Square[ALPHABET_LENGTH][ALPHABET_LENGTH] = {{' '}};
00049
00050     if ((Vigenere_Square[0][0]) == ' '){ // call first time only
00051         for (size_t i = 0; i < ALPHABET_LENGTH; i++) {
00052             for (size_t j = 0; j < ALPHABET_LENGTH; j++){
00053                 Vigenere_Square[i][j] = alphabet[(j + i) % ALPHABET_LENGTH];
00054             }
00055         }
00056     }
00057     // convert to integer to traverse array
00058     // the ascii code for each is
00059     // a = 97, A = 65;
00060     int scrt_char_n;
00061     int plain_char_n;
00062
00063     plainOrCipher_char = toupper(plainOrCipher_char);
00064     scrt_char = toupper(scrt_char);
00065
00066     // give ascii numbers
00067     if (isupper(scrt_char)){scrt_char_n = scrt_char - 65;} // A = 0, ... Z = 25
00068     if (isupper(plainOrCipher_char)) {plain_char_n = plainOrCipher_char - 65;}
00069
00070     // encrypt
00071     if((Encrypt)){
00072         return Vigenere_Square[scrt_char_n][plain_char_n];
00073     }
00074
00075     if(!(Encrypt)){
00076
00077 #ifdef DEBUG
00078         puts("Values supplied to function:");
00079         printf("scrt_char: %c\n", scrt_char);
00080         printf("plainOrCipher_char %c\n", plainOrCipher_char);
00081         puts("");
00082         puts("Converted to:");
00083         printf("scrt_char_n: %d\n", scrt_char_n);
00084         printf("plain_char_n: %d\n", plain_char_n);
00085 #endif
00086         for (size_t i = 0; i < ALPHABET_LENGTH; i++){
00087             if(Vigenere_Square[scrt_char_n][i] == plainOrCipher_char){
00088                 return Vigenere_Square[0][i];
00089             }
00090         }
00091     }
00092 }
00093
00098 int encrypt (char *plainToEncrypt, char *cipherTextArray, char *scrt_strng){
00099     int p_counter = 0, s_counter = 0;
00100
00101
00102     if(!checkKey(scrt_strng)){
00103         return EXIT_FAILURE;
00104     }
00105
00106     while (plainToEncrypt[p_counter] != '\0'){
00107
00108         if (!isalpha(plainToEncrypt[p_counter])){
00109             cipherTextArray[p_counter] = plainToEncrypt[p_counter];
00110             p_counter++;
00111             continue;
00112         }
00113
00114         cipherTextArray[p_counter] = getSubstitution(
00115                 scrt_strng[s_counter], plainToEncrypt[p_counter], true);
00116
00117         p_counter++; s_counter++;
00118
00119         if (scrt_strng[s_counter] == '\n'){
00120             s_counter = 0;
00121         }
00122     }
00123
00124     p_counter = 0;
00125
00126     return EXIT_SUCCESS;
00127 }
00128
00133 int decrypt (char *cipherToDecrypt, char *plainTextArray, char *scrt_strng){
00134     int p_counter = 0, s_counter = 0;
00135
00136     if(!checkKey(scrt_strng)){
00137         return EXIT_FAILURE;
00138     }
00139
00140     while (cipherToDecrypt[p_counter] != '\0'){
```

```
00141
00142         if (!isalpha(cipherToDecrypt[p_counter])){
00143             plainTextArray[p_counter] = cipherToDecrypt[p_counter]; // copy.. then move on
00144             p_counter++;
00145             continue;
00146         }
00147
00148         plainTextArray[p_counter] = getSubstitution (
00149                 scrt_strng[s_counter], cipherToDecrypt[p_counter], false);
00150
00151         p_counter++; s_counter++;
00152
00153         if (scrt_strng[s_counter] == '\n'){
00154             s_counter = 0;
00155         }
00156     }
00157
00158     p_counter = 0;
00159
00160     return EXIT_SUCCESS;
00161 }
00162
```

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 133 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 98 of file cipher.c.

3.4 cipher.h

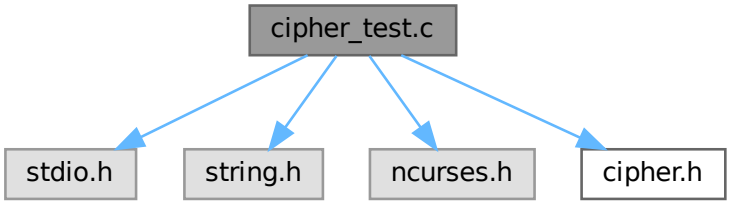
Go to the documentation of this file.

```
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 <string.h>
#include <ncurses.h>
#include "cipher.h"
```

Include dependency graph for cipher_test.c:



Functions

- int main (void)

3.5.1 Function Documentation

main()

```
int main (
            void )
```

Definition at line 9 of file cipher_test.c.

3.6 cipher_test.c

Go to the documentation of this file.

```
00001
00002 #include <stdio.h>
00003 #include <string.h>
00004 #include <ncurses.h>
00005 #include "cipher.h"
00006
00007 //#define DEBUG
00008
00009 int main(void){
00010     char title[] = "Vigenere Cipher Project";
00011     char encrypt_request[] = "Enter a sentence to encrypt - Maximum ";
00012     char decrypt_request[] = "Enter a sentence to decrypt - Maximum ";
00013     char secret_key_request[] = "Enter a secret key - Maximum ";
00014
00015     int ch;
00016
00017     char plainText[MAX_SENTENCE_LENGTH];
00018     char secretKey[MAX_KEY_LENGTH];
00019     char cipherText[MAX_SENTENCE_LENGTH];
00020
00021     initscr();
00022
00023     attron(A_REVERSE);
00024     addstr(title);
00025     attroff(A_REVERSE);
00026
00027     move(2,0);
00028
00029     printw("Select an option:\n");
00030     printw("1. Encrypt\n");
00031     printw("2. Decrypt (q to quit) \n");
00032
00033     while ( (ch = getch()) != 'q') {
00034
00035         if (ch == '1'){
00036             memset(plainText, 0, strlen(plainText));
00037             memset(cipherText, 0, strlen(cipherText));
00038             memset(secretKey, 0, strlen(secretKey));
00039
00040             printw("\nEncrypt\n");
00041             printw("%s %d characters:\n", encrypt_request, MAX_SENTENCE_LENGTH);
00042             refresh();
00043
00044             getnstr(plainText, MAX_SENTENCE_LENGTH);
00045
00046             printw("%s %d characters:\n" , secret_key_request, MAX_KEY_LENGTH);
00047             refresh();
00048             getnstr(secretKey, MAX_KEY_LENGTH);
00049
00050             refresh();
00051             encrypt(plainText, cipherText, secretKey);  // encrypt
00052
00053             printw("%s\n", cipherText);
00054             printw("(press any key to continue)");
00055
00056             getch();
00057
00058             clear();
00059             refresh();
00060
00061             printw("Select an option:\n");
00062             printw("1. Encrypt\n");
00063             printw("2. Decrypt (q to quit)\n");
00064         }
00065         if (ch == '2'){
00066             memset(plainText, 0, strlen(plainText));
00067             memset(cipherText, 0, strlen(cipherText));
00068             memset(secretKey, 0, strlen(secretKey));
00069
00070             printw("\nDecrypt\n");
00071             printw("%s %d characters:\n", decrypt_request, MAX_SENTENCE_LENGTH);
00072             refresh();
00073             getnstr(cipherText, MAX_SENTENCE_LENGTH);
00074
00075             printw("%s %d characters:\n" , secret_key_request, MAX_KEY_LENGTH);
00076             refresh();
00077             getnstr(secretKey, MAX_KEY_LENGTH);
00078
00079             refresh();
00080             decrypt(cipherText, plainText, secretKey); // decrypt
00081
00082             printw("%s\n", plainText);
```

```
00083             printw("(press any key to continue)");
00084
00085             getch();
00086
00087             clear();
00088             refresh();
00089
00090             printw("Select an option:\n");
00091             printw("1. Encrypt\n");
00092             printw("2. Decrypt (q to quit) \n");
00093         }
00094
00095         memset(plainText, 0, strlen(plainText));
00096         memset(cipherText, 0, strlen(cipherText));
00097         memset(secretKey, 0, strlen(secretKey));
00098
00099     }
00100
00101     endwin();
00102
00103     return 0;
00104 }
00105
```

3.7 Mainpage.dox File Reference

Index