**Introduction to Programming in C**

Assignment 4

Contents

[Part I – Code 1](#_Toc33655950)

[Part II – Problem Specification 5](#_Toc33655951)

[Part II – Analysis 6](#_Toc33655952)

[Part II – Design 6](#_Toc33655953)

[Part II – Implementation 7](#_Toc33655954)

[Part II – Testing and Verification 8](#_Toc33655955)

# Part I – Code

#include <stdio.h>

#include <stdlib.h>

#define MAX 500

int display\_menu(void);

void check\_exists(FILE \*checkFile);

void encrypt(FILE \*inputFile, char \*string3, FILE \*outputFile);

void decrypt(FILE \*inputFile, char \*string3, FILE \*outputFile);

void main(void)

{

FILE \*inputFile;

FILE \*keyFile;

FILE \*outputFile;

int option;

char input[MAX];

char key[MAX];

char output[MAX];

int i, j;

char key\_final[MAX];

char string3[MAX];

//Get option to encrypt or decrypt

option = display\_menu();

//Get input file name

printf("Enter input file name: ");

fflush(stdin);

scanf("%s", &input);

inputFile = fopen(input, "r");

//Check if file exists

check\_exists(inputFile);

//Get key file name

printf("\n\nEnter key file name: ");

fflush(stdin);

scanf("%s", &key);

keyFile = fopen(key, "r");

//Check if file exists

check\_exists(keyFile);

//Get output file name

printf("\n\nEnter output file name: ");

fflush(stdin);

scanf("%s", &output);

outputFile = fopen(output, "w");

//Make key longer

while((fgets(string3, sizeof(string3), keyFile)) != NULL)

{

for(i=0; string3[i] != 0; i++)

{

j=i;

}

for(i=i; i != MAX; i++)

{

string3[i] = string3[i - j];

}

}

//Switch based on encrypt or decrypt

switch(option)

{

case 1:

encrypt(inputFile, string3, outputFile);

break;

case 2:

decrypt(inputFile, string3, outputFile);

break;

}

//Close file

fclose(outputFile);

}

int display\_menu(void)

{

int option = 0;

do

{

printf("Select desired operation:");

printf("\n1 - Encrypt");

printf("\n2 - Decrypt");

printf("\nSelect an option: ");

fflush(stdin);

scanf("%d", &option);

}while(option != 1 && option != 2);

return option;

}

void check\_exists(FILE \*checkFile)

{

if(checkFile == NULL)

{

printf("\nCannot open file\nEnd program");

exit(1);

}

printf("\nFile found!");

}

void encrypt(FILE \*inputFile, char \*string3, FILE \*outputFile)

{

int y = 0, i, j;

char x;

char string[500], encryption[500];

while((fgets(string, sizeof(string), inputFile)) != NULL)

{

//MIght need to move y = 0 to here ON BOTH

for(i=0; i<MAX; i++)

{

if(string[i] == '\n')

{

i = 501;

fprintf(outputFile, "\n");

y = 1;

}

else

{

encryption[i] = string[i] + string3[i];

fprintf(outputFile, "%c", encryption[i]);

}

}

}

}

void decrypt(FILE \*inputFile, char \*string3, FILE \*outputFile)

{

int y = 0, i, j;

char x;

char string[500], encryption[500];

while((fgets(string, sizeof(string), inputFile)) != NULL)

{

for(i=0; i<MAX; i++)

{

if(string[i] == '\n')

{

i = 501;

fprintf(outputFile, "\n");

y = 1;

}

else

{

encryption[i] = string[i] - string3[i];

fprintf(outputFile, "%c", encryption[i]);

}

}

}

}

# Part II – Problem Specification

The problem encountered is that a method is required to aid in the safe transfer of a file that, due to its nature, can only be sent in plain format. It has been determined that the solution is to create, utilizing the c language, an encryption algorithm that can accomplish this by reading in the text document containing the information to be encrypted and a numerical key from a separate document. The program should be able to get the information from the user regarding what the file is called and then read this selected file into the program. This should be repeated for the file containing the key. If the file entered does not exist, then the program should exit. Furthermore, the output file name should also be selected by the user. If a file does not already exist with this name, then it will create this file and write to it. If a file already exists with this name, then it will overwrite it.

The program will have a maximum character length of 500 however this should be easily changeable. If the numerical key is shorter than the 500-character limit, then it should be changed to repeat itself to fill that space. The program will then use the numerical key to alter the text read from the original file. This this will result in a different set of characters than originally entered. For instance, is the message to be encrypted is “HELLO” and the key is “1234” then the result should be:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| H | E | L | L | O |
| 1 | 2 | 3 | 4 | 1 |
| I | G | O | P | P |

Each number of the key is added to the message to create this altered message. This should then be output to the file indicated by the user.

The program should

# Part II – Analysis

Inputs:

* option (int) – This will hold a 1 or a 2 integer to indicate the users option depending on if they want to encrypt a file or decrypt it.
* inputFile (char) – This is the file that must be encrypted or decrypted depending on the user’s option. This will be read to the program.
* keyFile (char) – This is the file that contains the key that is used to alter the message. This will be read to the program.

Outputs:

* outputFile (char) – This is the file that the program will write the completed message to.

Relevant aspects

* Encryption – The encryption process will be completed by taking the character that it is dealing with and adding the corresponding number from the key.
* Decryption – The decryption process will be completed by taking the character that it is dealing with and taking away the corresponding number from the key.

# Part II – Design

Get option to encrypt or decrypt.

* Do while nether option is valid (valid is 1 or 2).
  + Print instructions to user.
  + Use fflush before getting input.
  + Get option.
* Return option to main function.

Get input file name.

* Print instructions to user.
* Use fflush before getting input.
* Get name of input file from user.

Check if input file exists.

* Function takes in inputFile.
  + If file taken in is NULL, then print info to user and exit program.
  + Otherwise tell user that the file has been found (this does need to be contained in an else condition because if the first is true then the program will have already exited).

Get output file name from user.

* Print instructions to user.
* Use fflush before getting input.
* Get name of output file from user.

Check if output file exists.

* Function takes in outputFile.
  + If file taken in is NULL, then print info to user and exit program.
  + Otherwise tell user that the file has been found (this does need to be contained in an else condition because if the first is true then the program will have already exited).

Make key longer.

* Repeat the key until it is the required key in the right order

Switch between encrypt and decrypt

* Encrypt
  + Take in the input file, string3 and outputFile
  + For each character in each line add the character in the key
  + Print character to new file
* Decrypt
  + Take in the input file, string3 and outputFile
  + For each character in each line take away the character in the key
  + Print character to new file

Close the file

# Part II – Implementation

int display\_menu(void);

This is for the get option to encrypt or decrypt function. It gets the option from the user depending on what they type of operation they want to perform and then returns it to the main function to be used in the switch statement later to determine if it use the encrypt or decrypt function.

void check\_exists(FILE \*checkFile);

This function takes in the file that the user has selected and checks to confirm that it exists. If it does not then the program will exit.

void encrypt(FILE \*inputFile, char \*string3, FILE \*outputFile);

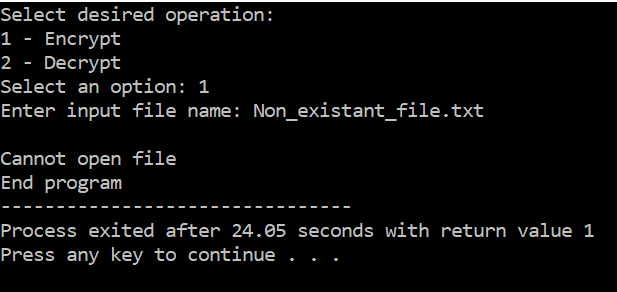
This function is used to encrypt the message file, using the string that has come from the key file, and output it to the new file that the program has either created or overwritten.

void decrypt(FILE \*inputFile, char \*string3, FILE \*outputFile);

This function is used to decrypt the message file, using the string that has come from the key file, and output it to the new file that the program has either created or overwritten.

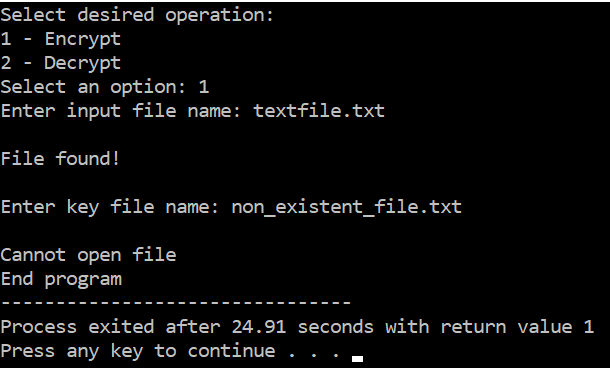
# Part II – Testing and Verification

Test 1: Entering non-existing input file



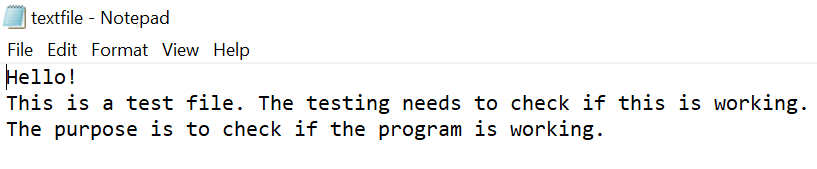
This test suggests this part works correctly. Firstly, the program seems to accept the option. Then when a non-existent file name was entered, the program was able to recognize this and exit.

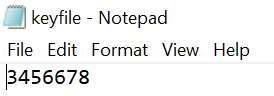
Test 2: Entering non-existing key file

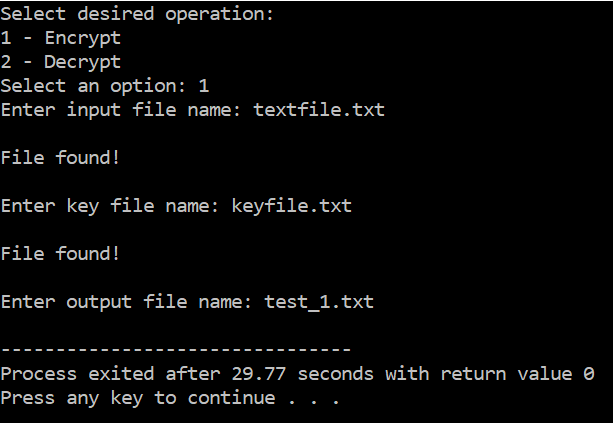


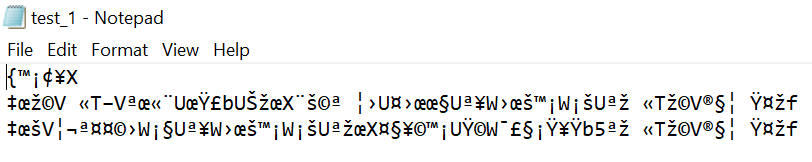
This test also seems to be successful. First it accepted the option correctly, then it was able to find the file that did exist, and finally it was able to recognize that the key file did not exist and exit.

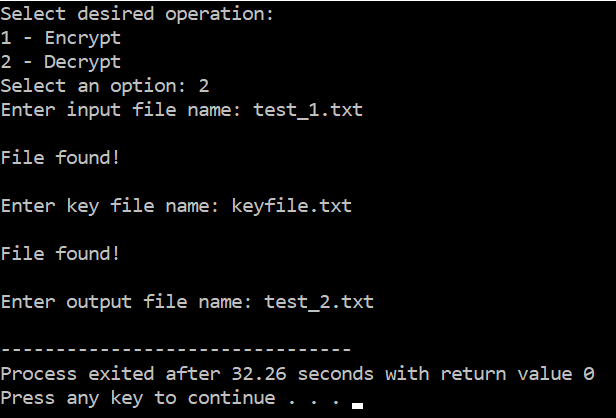
Test 3: Encrypting and decrypting a file

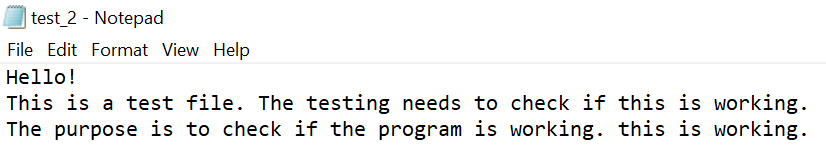










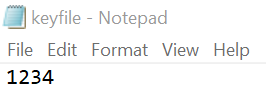
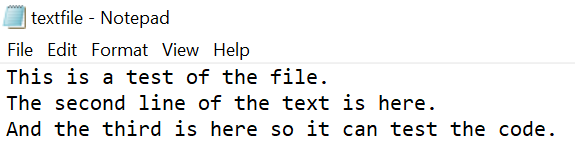


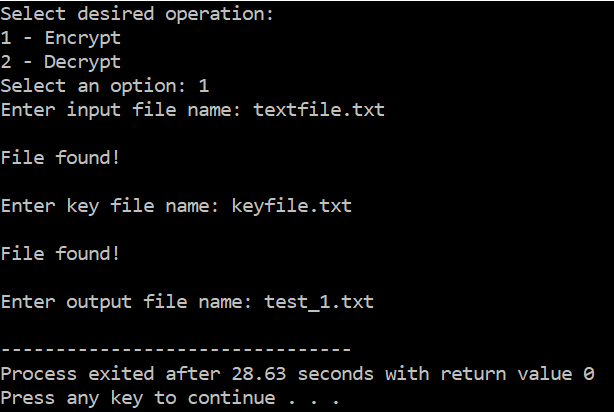
This test case did not work as intended. The first issue that appears is that the encrypted file does not encrypt as intended. This is likely because it is adding a char and a char rather than taking the key as an integer. This means that when the program adds a character from the key it is instead adding the code associated with that number rather than adding it as an integer itself. This issue is relatively minor because it still allows the code to be encrypted and decrypted back into its original form.

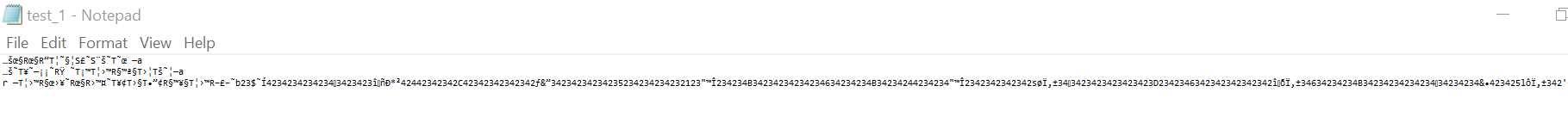
The most significant issue with this test case is that the final line contains writing from the line above it. Rather than ending, it has continued to print to the file what should not be there. Whilst the program does decode the message, it does not do it accurately to what it was before and adds this to the end. This issue appears to be present when the final line of the original message is shorter than the text above it.

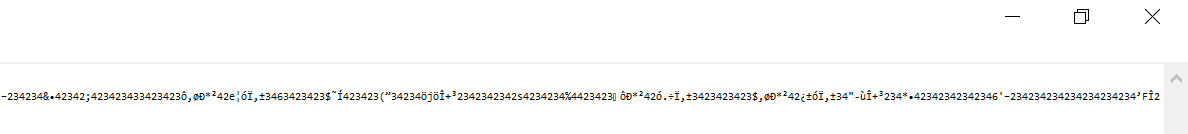
This shows that the program does not work as intended.

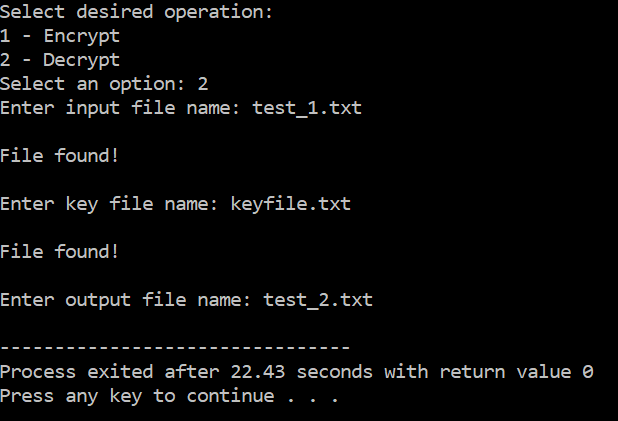
Test 4: Encrypting and decrypting a file

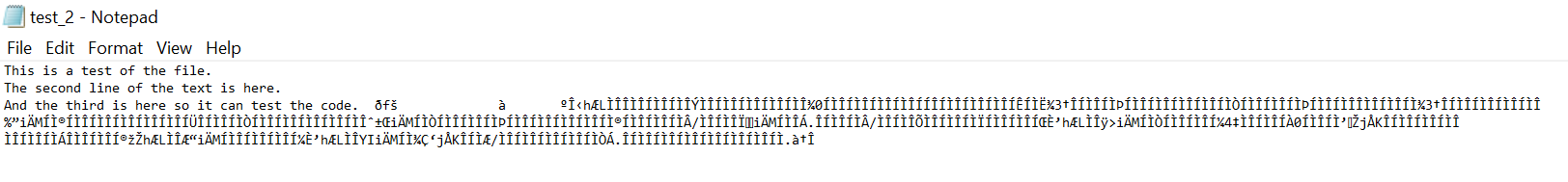












This test case also shows issues. The same issue, that was present in the previous test case, is present in this test case due to each part of the key being a char rather than an integer. However, this still does not stop it being encoded and decoded, it just results in it working slightly differently.

The main issue is that after the final line is prints a long number of characters. In the encrypted text file this is present as a very long line of characters that is difficult to show. In the final decrypted text file, the original message is decrypted back however these additional characters are also included after the original message. This issue seems to be present when the last line of the text in the original message is longer than the line above it.

This shows that the program does not work as intended.