CSO: Recitation 4

Goktug Saatcioglu

09.26.2019

1 I/O in C

Today we will learn about I/O in C. The task is to implement a program in C that encrypts and decrypts a file using a Caesar Cipher. The user will provide what service it would like, `e` for encrypt and `d` for decrypt, through the standard input along with a file name that is in the current directory the file is running in. Furthermore, the user must provide a value `k` that will be used for the encryption/decryption process. We begin by describing what a Caeser Cipher is.

The Caesar cipher is one of the earliest known and simplest ciphers. It is also know an as a substitution cipher where ach letter encountered in a piece of plaintext is shifted by a certain number of places down the alphabet. If `k`, the shift, is 1 then A becomes B, B becomes C and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals.

Our program will encrypt and decrypt using the following scheme:

- If a character c is a number from 0 to 9 then c will be shifted k places to the right (for example if c is 0 and k is 1 we get 1) for encryption and k places to the left for decryption. c wraps around from 9 to 0 and so forth.
- If a character c is an upper-case letter from A to Z then c will be shifted k places to the right for encryption and k places to the left for decryption. c wraps around from Z to A and so forth.
- If a character c is an lower-case letter from a to z then c will be shifted k places to the right for encryption and k places to the left for decryption. c wraps around from z to a and so forth.

Let's begin by writing some logic into a file named `main.c` that will read `k`, `file_name` and `(e)ncrypt` or `(d)ecrypt` from the standard input. As you may recall from class, we will require the `<stdio.h>` library and the `scanf` function to get all the information we need. Try implementing this functionality yourself and then see below for one possibility. We would like to modularize our program as much as possible so we create two new files, one called `input_manager.c` that will help us manager I/O and another called `input_manager.h` that will be the header file for the implementation file. One possible solution is given below.

Listing 1: input_manager.h

```
#pragma once

void initialize(int*, char**, char*);
```

Listing 2: input_manager.c

```
#include <stdio.h>

void initialize(int* k, char** file_name, char* encrypt_input) {
  printf("Hello! Welcome to the Caeser Cipher program.\n");
  printf("\nWould you like to (e)ncrypt or (d)crypt: ");
  scanf("%c", encrypt_input);
  printf("What is the number of shifts in your scheme (k): ");
  scanf("%d", k);
  printf("Name of the file you would like to process: ");
  scanf("%s", *file_name);
  printf("\nPlease wait while we process your input.\n");
}
```

Listing 3: main.c

```c
#include <stdio.h>
#include "input_manager.h"

int main() {

  int k;
  char* file_name;
  char encrypt_input;

  initialize(&k, &file_name, &encrypt_input);

  if (encrypt_input != 'e' && encrypt_input != 'd') {
    printf("Something went wrong while you were inputting your request.\n");
    printf("You may only enter \'e\' for encryption and \'d\' for decryption.\n");
    printf("Please try again.");

    return 2;
  }

  return 0;
}
```

Let's now implement logic to open a file. We need to use the `fopen` function to open a file and have a `FILE` pointer point to the file. Of course, we need to check whether we have actually opened the file before being able to use it. In that case, `FILE` would be a `NULL` pointer. We should also open a new file that we will write to so that we can save the output somewhere. This file should always be called `RESULT`. The solution now becomes as follows.

Listing 4: input_manager.h

```c
#pragma once

void initialize(int*, char**, char*);
void open_file(FILE**, char**, char*);
```

Listing 5: input_manager.c

```c
#include <stdio.h>

void initialize(int* k, char** file_name, char* encrypt_input) {
  printf("Hello! Welcome to the Caeser Cipher program.\n");
  printf("\nWould you like to (e)ncrypt or (d)crypt: ");
  scanf("%c", encrypt_input);
  printf("What is the number of shifts in your scheme (k): ");
  scanf("%d", k);
  printf("Name of the file you would like to process: ");
  scanf("%s", *file_name);
  printf("\nPlease wait while we process your input.\n");
}
```

```c
void open_file(FILE** file_pointer, char** file_name, char* file_mode) {
  *file_pointer = fopen(*file_name, file_mode);
}
```

Listing 6: main.c

```c
#include <stdio.h>
#include <string.h>
#include "input_manager.h"

#define RESULT_FILE_NAME "RESULT"

int main() {

  int k;
  char* in_file_name;
  char encrypt_input;
  initialize(&k, &in_file_name, &encrypt_input);

  if (encrypt_input != 'e' && encrypt_input != 'd') {
    printf("Something went wrong while you were inputting your request.\n");
    printf("You may only enter \'e\' for encryption and \'d\' for decryption.\n");
    printf("Please try again.");

    return 2;
  }

  FILE* in_file;
  open_file(&in_file, &in_file_name, "r");

  if (in_file == NULL) {
    printf("No file named %s found. Please try again.\n", in_file_name);

    return 3;
  }

  FILE* out_file;
  char* out_file_name = RESULT_FILE_NAME;
  open_file(&in_file, &out_file_name, "w");

  return 0;
}
```

Now we need to actually implement our logic for encrypting and decrypting a file. We need to read a file character by character, check if the current character c is a number, lower-case letter or upper-case letter, shift it according to the rules and then write the new character c' to the new file. We can shift characters by using the ASCII table and applying modular arithmetic. For example, 0 is 48 in ASCII and 9 is 57 in ASCII. A shift forward by k for character c would become $(((c - 48) + k) \mod (57 - 48)) + 48$ is one way of calculating the new character. Of course, there are multiple ways of doing this. How you wish to achieve the encryption/decryption is up to you. You will want your cipher logic to be in a new `.c` file, say `cipher_manager.c`. Furthermore, we would like our input manager to actually take care of reading and writing to a file. Finally, if a file is succesfully encrypted/decrypted we should return 1 for true and otherwise 0 for false to the main function. Give it a go and then check your solution against what is given below.

Listing 7: input_manager.h

```h
#pragma once
#include <stdio.h>

void initialize(int*, char**, char*);

void open_file(FILE**, char**, char*);
void close_file(FILE**);

int read_char_from_file(FILE**);
int write_char_to_file(FILE**, int);
```

Listing 8: input_manager.c

```c
#include <stdio.h>

void initialize(int* k, char** file_name, char* encrypt_input) {
  printf("Hello! Welcome to the Caeser Cipher program.\n");
  printf("\nWould you like to (e)ncrypt or (d)crypt: ");
  scanf("%c", encrypt_input);
  printf("What is the number of shifts in your scheme (k): ");
  scanf("%d", k);
  printf("Name of the file you would like to process: ");
  scanf("%s", *file_name);
  printf("\nPlease wait while we process your input.\n\n");
}

void open_file(FILE** file_pointer, char** file_name, char* file_mode) {
  *file_pointer = fopen(*file_name, file_mode);
}

void close_file(FILE** file_pointer) {
  fclose(*file_pointer);
}

int read_char_from_file(FILE** input_file) {
  return fgetc(*input_file);
}

int write_char_to_file(FILE** output_file, int c) {
  return fputc(c, *output_file);
}
```

Listing 9: cipher_manager.h

```h
#pragma once
#include <stdio.h>

int encrypt(FILE**, FILE**, int);
int decrypt(FILE**, FILE**, int);
```

Listing 10: cipher_manager.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "input_manager.h"

#define NUM_DIFF 10
#define LETTER_DIFF 26

int encrypt(FILE** input_file, FILE** output_file, int k) {
  int curr ;

  while ((curr = read_char_from_file(input_file)) != -1) {
    int res;

    if (curr >= 48 && curr <= 57) {
      res = ((curr - 48 + k) % NUM_DIFF) + 48;
    } else if (curr >= 65 && curr <= 90) {
      res = ((curr - 65 + k) % LETTER_DIFF) + 65;
    } else if (curr >= 97 && curr <= 122) {
      res = ((curr - 97 + k) % LETTER_DIFF) + 97;
    } else {
      res = curr;
    }

    if (write_char_to_file(output_file, res) == -1)
      return 0;
  }

  return 1;
}

int decrypt(FILE** input_file, FILE** output_file, int k) {
  int curr ;

  while ((curr = read_char_from_file(input_file)) != -1) {
    int res;

    if (curr >= 48 && curr <= 57) {
      res = (abs((curr - 48 - k)) % NUM_DIFF) + 48;
    } else if (curr >= 65 && curr <= 90) {
      res = (abs((curr - 65 - k)) % LETTER_DIFF) + 65;
    } else if (curr >= 97 && curr <= 122) {
      res = (abs((curr - 97 - k)) % LETTER_DIFF) + 97;
    } else {
      res = curr;
    }

    if (write_char_to_file(output_file, res) == -1)
      return 0;
  }

  return 1;
}
```

All that remains is to finish to add extra logic into the main function to either encrypt or decrypt a file and then close all files before exiting the progra. One possible solution is given below.

Listing 11: main.c

```c
#include <stdio.h>
#include <string.h>
#include "input_manager.h"
#include "cipher_manager.h"

#define RESULT_FILE_NAME "RESULT"

int main() {

  int k;
  char* in_file_name;
  char encrypt_input;
  initialize(&k, &in_file_name, &encrypt_input);

  if (encrypt_input != 'e' && encrypt_input != 'd') {
    printf("Something went wrong while you were inputting your request.\n");
    printf("You may only enter \'e\' for encryption and \'d\' for decryption.\n");
    printf("Please try again.");

    return 2;
  }

  FILE* in_file;
  open_file(&in_file, &in_file_name, "r");

  if (in_file == NULL) {
    printf("No file named %s found. Please try again.\n", in_file_name);

    return 3;
  }

  FILE* out_file;
  char* out_file_name = RESULT_FILE_NAME;
  open_file(&out_file, &out_file_name, "w");

  int res;
  if (encrypt_input == 'e') {
    res = encrypt(&in_file, &out_file, k);
  } else {
    res = decrypt(&in_file, &out_file, k);
  }

  close_file(&in_file);
  close_file(&out_file);

  if (res == 0) {
    printf("Something went wrong while encrypting or decrypting your file.\n");
    printf("Please try again later.\n");
```

```
    return 4;
  }

  printf("Your operation was successfully completed. See %s for the output.\n", RESULT_FILE_NAME);
  return 0;
}
```

This completes our exercise. As a bonus challenge you can try encrypting the whole ASCII table or reading the file using `fscanf` and writing to the file using `fprintf`. You could also try a different approach where you load the contents of a file to an array first and work on that array instead. Finally, you can also try to manage the output file naming scheme better by trying more sophisticated string manipulation. For example, you could concatenate a unique identifier to the end of the file name and have that be the output file's name. Note that these are all just bonus exercises to get you even more familiar with C programming.