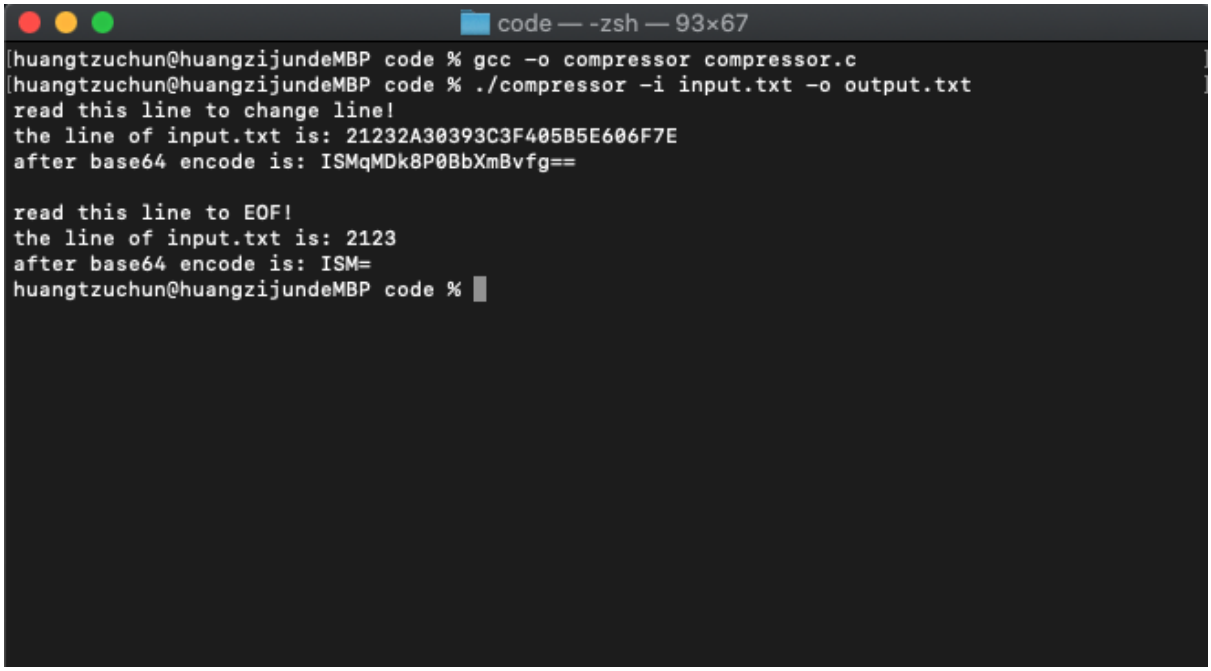
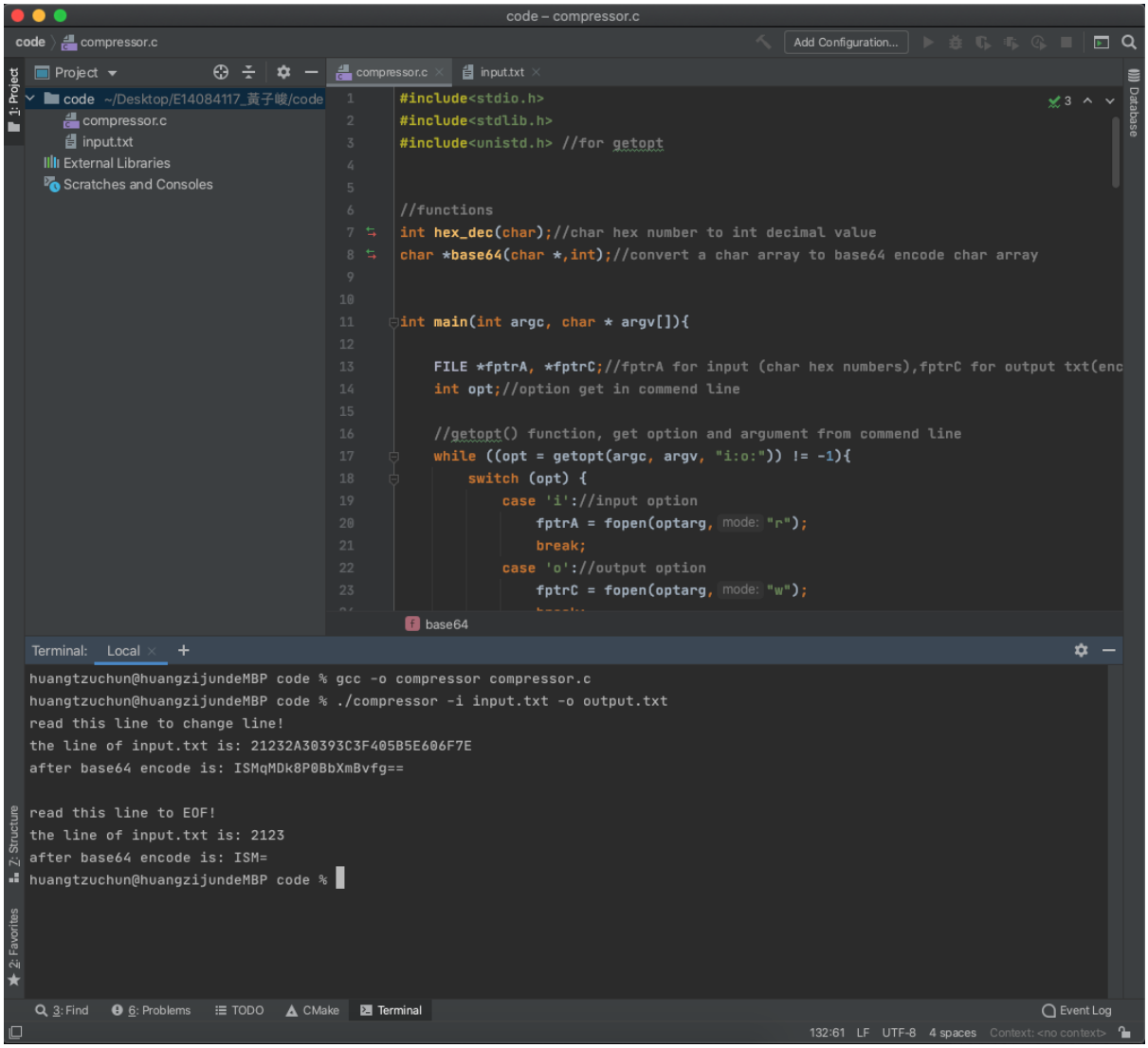
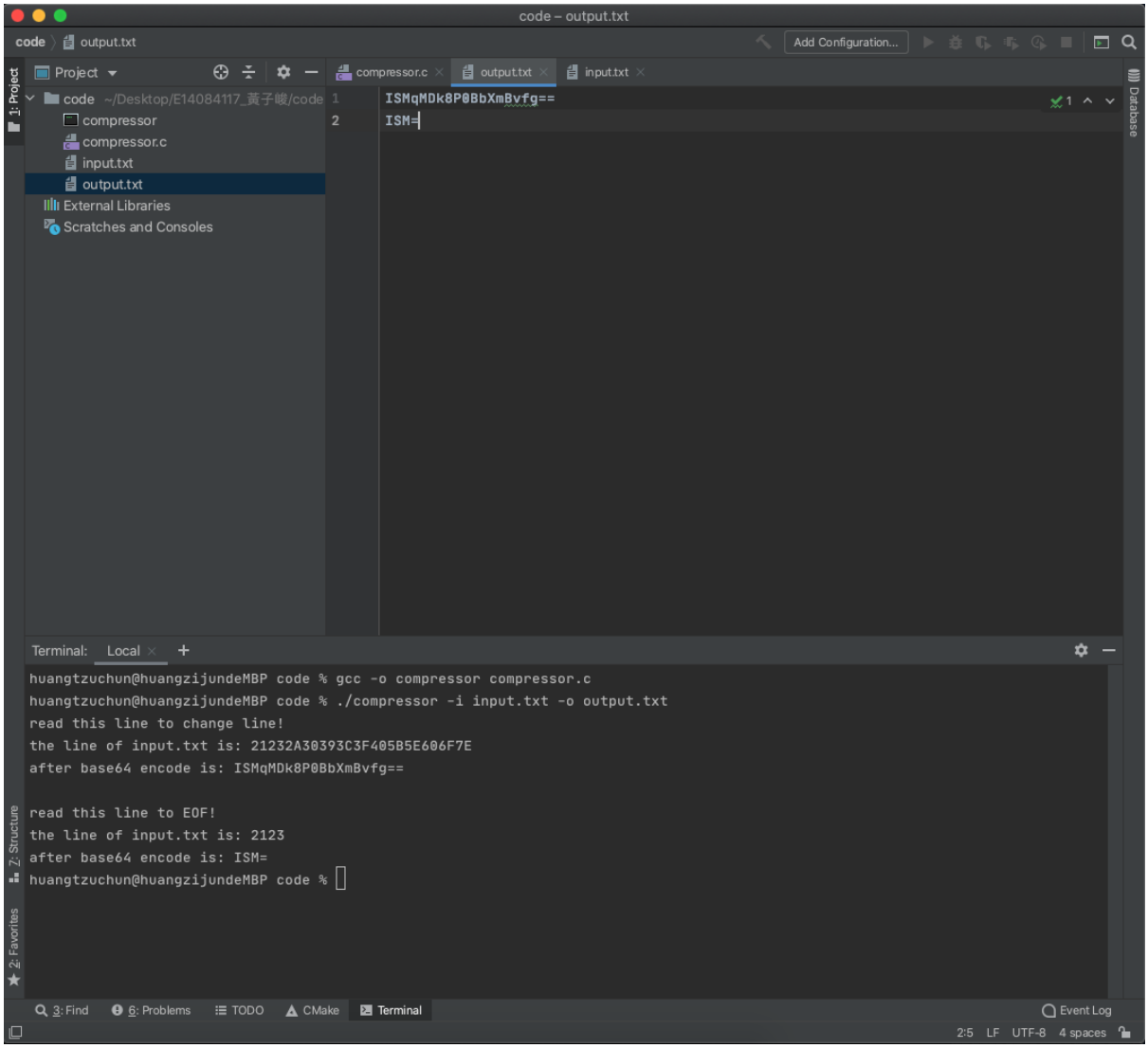
Readme

(1) Result screenshot:

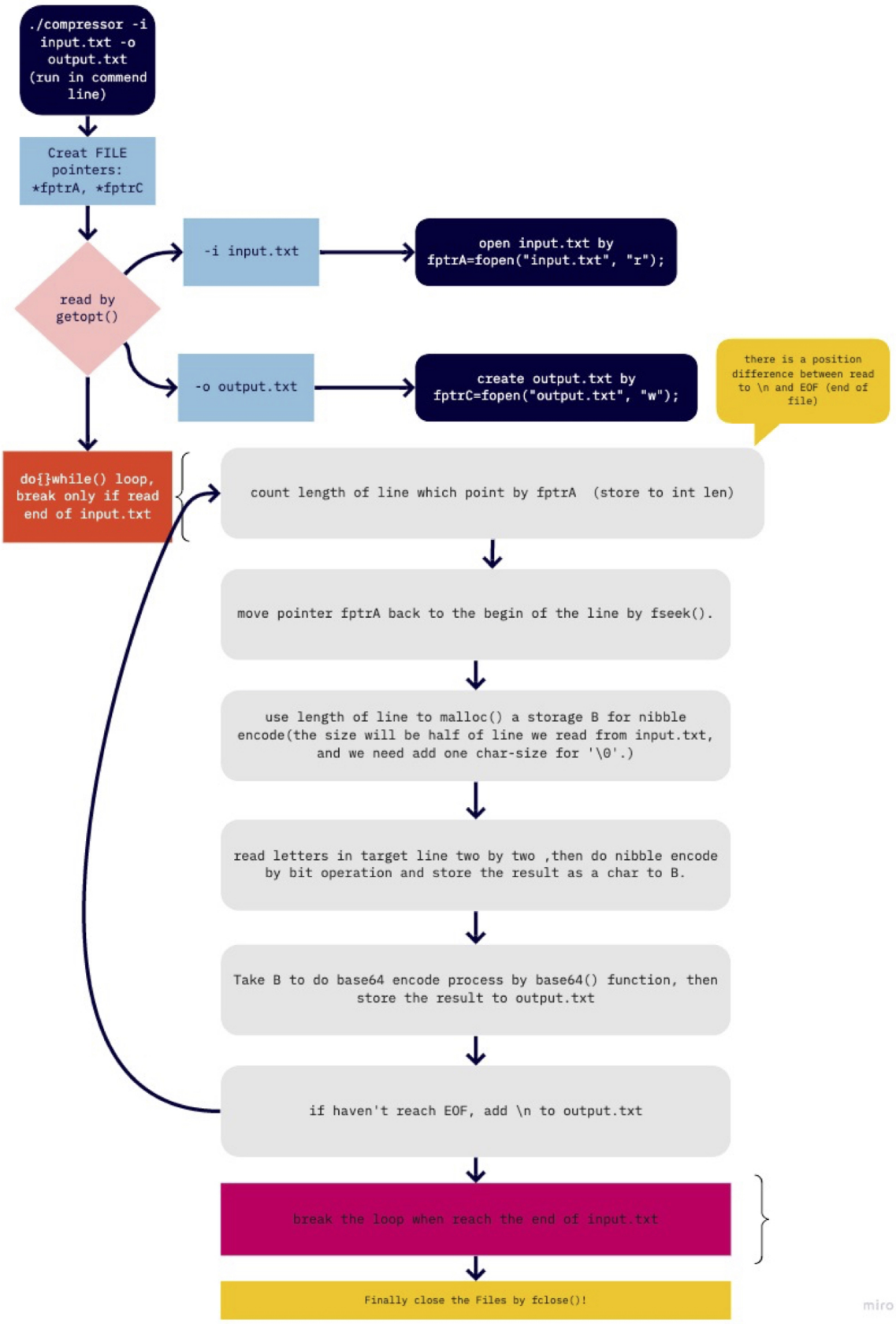



code – compressor.c
code compressor.c
Add Configuration...
Project
code ~/Desktop/E14084117_黃子峻/code
compressor.c
input.txt
External Libraries
Scratches and Consoles
compressor.c
input.txt
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h> //for getopt
//functions
int hex_dec(char);//char hex number to int decimal value
char *base64(char *,int);//convert a char array to base64 encode char array
int main(int argc, char * argv[]){
FILE *fptrA, *fptrC;//fptrA for input (char hex numbers),fptrC for output txt(enc
int opt;//option get in commend line
//getopt() function, get option and argument from commend line
while ((opt = getopt(argc, argv, "i:o:")) != -1){
switch (opt) {
case 'i'://input option
fptrA = fopen(optarg, mode: "r");
break;
case 'o'://output option
fptrC = fopen(optarg, mode: "w");
base64
Terminal: Local
huangtzuchun@huangzijundeMBP code % gcc -o compressor compressor.c
huangtzuchun@huangzijundeMBP code % ./compressor -i input.txt -o output.txt
read this line to change line!
the line of input.txt is: 21232A30393C3F405B5E606F7E
after base64 encode is: ISMqMDk8P0BbXmBvfg==
read this line to EOF!
the line of input.txt is: 2123
after base64 encode is: ISM=
huangtzuchun@huangzijundeMBP code %
3: Find
6: Problems
TODO
CMake
Terminal
Event Log
132:61 LF UTF-8 4 spaces Context: <no context>



Note this program is running successful when input.txt is set to CRFL on clion and terminal of MacOS.(The result maybe different on vscode).

(2) Program architecture:



(3) Program functions:

`int hex_dec(char hex)`

Parameters

- `hex` - This is the char type hex value(0~F).

Return Values

- This function returns int type value (0~16) which turned by input char hex value(0~F).
- example- input: 0 → output : 0; input: A → output : 10;

`char *base64(char *str, int len_in)`

Parameters

- `str` - This is the string (char type letters,e.g. "abc") which you want to do base64 encode.
- `len_in` - This is the length of input string, with int type.

Return Values

- This function returns a string (char type letters) which encoded by base64 process.
- example- input: 0 → output : 0; input: A → output : 10;

`getopt(int argc, char **argv, const char optstring)`

Parameters

- `argc` - the argument count of main function
- `argv` - the argument vector of main function
- `optstring` - the options and parameter which you want to get in commend line.(in this program we set it as "i:o:" for option -i (for input) and option -o (for output), both can fallow by a parameter (file name).

Return Values

- This function returns option which read in commend line as a char type.
- This function can set parameter which read in commend line to variable `optarg`.

(4) How I design my program:

> 💡 For compress input.txt by base64 encoding, there are some problems I need to solve, and here are my solutions.

- Took data in input.txt than encode it by nibble and store it.

1. Because I want to store nibble results line by line to dynamic char array which can put into function base64(), first need to count the length of the target line.
2. We know that length of lines are even numbers and two units of nibble data are 8 bits, so I want to process the target line by two letters a time. First, change the char type hex number to its int type decimal value by my function `hex_dec()`. Second, do << `4-bit` operation to first letter's decimal value in two-letter pair, then use `|` operation to combine the first and second letter's decimal value and assign the result to an 8 bit-size char. The third is to put the result from the last step to the right place in our dynamic storage.
3. `00001111<<4==11110000` ;`11110000|00001111==11111111`;

- Took data after nibble and encode it by base64.

1. My base64() function can take a char array as an input and return the char array which is the result of input after base64 encode.
2. For dynamic allocation, we need to compute the length of input and output. The 3 letters(8*3=24bits) in input can turn in to 4 base64 data (6*4=24bits). By this specialty, we need to process the input letters four by four. The steps are:
    1. Take the first 6 bits from the first char.
    2. Take last 2 bits from the first char and first 4 bits from the second char.
    3. Take last 4 bits from the second char and first 2 bits from the third char.
    4. Take last 6 bits from the fourth char.

3. The length of the output is `(length of input/3*4)` *if the length of input letter is not a multiple of 3, otherwise set length of output to* `(length of input)/3*4+4` (`/3` operation only take the integer part).
4. If the length of the input letter is not a multiple of three we will need to add numbers of `'='` at the end of the line. (`(length of input%3)==1`, add two `'='`; `(length of input%3)==2`, add one `'='`)Otherwise, if the length of the input is a multiple of 3, there's no need to add `'='` .

- Dealing with line changing in input.txt and output.txt

1. The main do()while{} loop wants to process data line by line, so when doing nibble encode to the end of line in `input.txt,` judge the end of the line can be read as '\n' or EOF. If read to '\n', add '\n' to `output.txt`.

- Used this command `./compressor -i input.txt -o output.txt` to run the program.

1. By using getopt() function, create two option -i for input and -o for output. Then we can take arguments `input.txt` ant `output.txt` by use `optarg`.
2. in the switch(){} for option, assign `fptrA = fopen(optarg,"r")` and `fptrC = fopen(optarg,"w")` for read `input.txt` and create `output.txt`.