CIS 2107 Lab 3 (100 points)

This lab has 2 parts: Reading and Writing Files (40 points) and Encryption (60 points)

Part 1: Reading and Writing Complete Files (40 points)

The first part of the lab is to write a program to read the complete contents of a file to a string. This code will be used in subsequent coding problems. You will need 3 functions: main(), read_file() and write_file(). The main function contains the driver code. The read_file() function reads the complete contents of a file to a string. The write_file() writes the complete contents of a string to a file. The read_file() will require:

Open the file for reading **binary**
Calculating the size of a file using fseek(), ftell() and rewind()
Allocating memory to read the file to a string using malloc()
Reading the files contents to the allocated string using fread()
Close the file

To get the length of a file and move the file pointer back to the beginning:

```
// Get size of file
fseek(file, 0, SEEK_END);
unsigned long size = (unsigned long) ftell(file);
// Go to beginning of file
rewind(file);
```

Remember that when opening a file or allocating memory to check that the operation was successful. Both malloc() and fopen() return NULL when they fail. Also make sure to free all allocated memory when it is no longer needed.

The suggested function header for read_file is:

```
char* read_file(unsigned long *size, char *file_name)
```

This function accepts a reference to size and a char array with the filename, and returns the allocated char array containing the contents of the file. Size is passed by reference to record the number of bytes in the file. Writing the file will require:

Open the file for writing **binary**
Write the string to file using fwrite()
Close the file

The suggested function header for write_file is:

```
int file_write(unsigned long size, char *output, char *file_name)
```

This function accepts the size of the char array to be written to file, the char array to be written and the filename, and returns the number of bytes written to file. There is useful information about the C functions above at:

http://www.cplusplus.com/reference/

and

https://www.tutorialspoint.com/c_standard_library/index.htm

The main function should free() allocated memory before returning. Test your program by reading and writing a few files. Make sure to give the written file different names than the read files so you can compare them.

Part 2: One Time Pad Encryption (60 points)

One Time Pad (OTP) encryption, when implemented properly, cannot be hacked by cryptanalysis. This method uses the clear text (message to be encrypted), a random key of characters (the same size of the clear text) and the exclusive OR (XOR) operator. The table below contains the truth tables for bitwise operators. These operators work bit-by-bit on C data.

AND (&)	0	1
0	0	0
1	0	1

OR (|)	0	1
0	0	1
1	1	1

XOR (^)	0	1
0	0	1
1	1	0

The table below shows the result of encrypting the clear text A with the random key * using the bit-by-bit XOR bitwise operator. Each bit position in A is XORed with its respective bit in *, producing cipher text k.

	char	int	hex	bin
Clear text	A	65	41	01000001
				XOR
Rand key	*	42	2A	00101010
Cipher text	k	107	6B	01101011

The clear text can be recovered from the cipher text by using the cipher key and the XOR operator as shown in the table below.

	char	int	hex	bin
Cipher text	k	107	6B	01101011
				XOR
Rand key	*	42	2A	00101010
Clear text	A	65	41	01000001

There is an article about OTP at:

https://en.wikipedia.org/wiki/One-time_pad

This project will require the use of the read_file() and write_file() from Part 1 and the following functions:

1. main function with a menu including 3 choices: encrypt file, decrypt file and exit program.
2. make_rand_key(length of key, key) generates and returns a fixed length string of random chars.
3. encrypt(clear file, key file, cipher file) reads a clear text file using the read_file function, generates a random key using make_rand_key(), perform the clear text XOR key text, byte-by-byte, to produce the cipher text string and write both the random key and cipher text strings to file using the write_file function.
4. decrypt(key file, cipher file, clear file) reads the key and cipher files to strings, uses the cipher text XOR key to recover the original clear text string and writes the clear text to a file using the write_file function (make sure to use a different clear text file name than the original file so you can compare the results)

The menu in main() should look like:

```
Encrypt a file:    1
Decrypt a file:    2
Exit:              3
Enter a choice:
```

I suggest using a loop in main to show the menu, get a choice from the user and a switch statement to execute the chosen function. For the first two choices, the necessary file names should be read by main() and passed to encrypt() and decrypt(). To generate the random key, the length of the key and a char array pointer are passed to make_rand_key(). After make_rand_key() completes, the char array contains the randomly selected chars. When the user chooses to exit the program by entering 3, the loop should exit.

The make_rand_key() accepts the length of the random key char array and a pointer to this array. This function uses the srand(time(NULL)) function to seed the rand() function and the rand() function to get random chars. The rand() function accepts no argument and returns an integer value between 0 and RAND_MAX (32,767 on most Intel-based computers). The return from rand must be scaled between 0 and 255 and explicitly castes as a char before being added to the char array. A sample function call to make_rand_key is:

make_rand_key(length, key); // Where len > zero, generates and returns a random key of length chars.

The encrypt() function accepts the clear text, random key and cipher text file names as arguments. It reads the clear text file into a string using the read_file function, generates a random key string, performs XOR on clear text and random key, producing the cipher text string, and writes the random key and cipher text strings to file using the write_file function. Sample function call for encrypt():

encrypt(clear_file, key_file, cipher_file); // Where the three arguments are strings containing filenames.

The decrypt() function accepts the random key, cipher text and message (should be exactly same as clear text) text file names as arguments. It reads the random key and cipher text files into strings using the read_file function, performs XOR on cipher text and key, producing the message text string (make sure to have an extra char in the message text string for the null char), and writes the message text string to file using the write_file function. Sample function call for decrypt():

decrypt(key_file, cipher_file, message_file); // Where the three arguments are strings containing filenames.

Note: You must check if memory is successfully allocated and that files are opened successfully before using pointers to these objects. If a memory allocation or file open is unsuccessful, the pointer will be NULL. If a NULL pointer is found, you should provide an appropriate error message and terminate the program. You can test that the open file check works by passing a filename that does not exist. Also, make sure to free() any allocated memory before any function returns if it's no longer needed. Failing to free allocated memory can lead to a memory leak and is poor programming practice.