CSE 3100 Systems Programming	Fall 2025
Homework #5	

Do not forget to submit your .c code to gradescope. A Makefile is provided.

Exercise 1. (100 points) Breaking the code

In a hacking competition, our team breaks into the adversary's computer and finds the following files.

```
$ ls
encrypt.c           encrypted-message2  encrypted-message4  encrypted-message6
encrypted-message1  encrypted-message3  encrypted-message5  encrypted-message7
```

We cannot see the content of the seven encrypted files but we are able to see the content of file encrypt.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>

#define MAX 10240
//encrypt the message and write the result to file fd
void encrypt(char *message, unsigned char key, unsigned char shift, int fd)
{
    int len = strlen(message);
    printf("len = %d\n", len);
    int msg[len];

    for(int i = 0; i<len; i++)
        msg[i] = (message[i] << shift) ^ key;
    write(fd, msg, len*sizeof(int));
    printf("%s\n", (char *)msg);
}

int main(int argc, char *argv[])
{
    if(argc != 5)
    {
        printf("Usage: %s plain-text-file encrypted-file key shift\n", argv[0]);
        return -1;
    }

    char message[MAX];
    int fd = open(argv[1], O_RDONLY);
    if(fd < 0)
    {
        printf("Cannot open file %s\n", argv[1]);
        return -1;
    }
    int len = read(fd, message, MAX);
```

```
    close(fd);
    assert(len > 0);

    //note how we open the file
    fd = open(argv[2], O_WRONLY | O_TRUNC | O_CREAT, 0600);
    if(fd < 0)
    {
        printf("Cannot open the file\n");
        return -1;
    }
    int key = atoi(argv[3]);
    int shift = atoi(argv[4]);
    assert(key >= 0 && key <= 255);
    assert(shift >=0 && shift <= 24);
    encrypt(message, key, shift, fd);
    //remember to close the file
    close(fd);
    return 0;
}
```

We believe these seven files are encrypted using the above code. Now we need to break the code: decrypt the seven files to see their original content.

From the code we can see that each character in the original message is converted to an integer by applying bit shifting (`<<`) and then an exclusive or (`^`) operation with a key. The number of shifts and the key are not known, since they are passed as command line arguments. We will use a brute-force method to try all combinations of shift and key values and see which combination reveals the original message.

To break the code, we use a dictionary to check the decrypted text. For example, we can count the number of words in the decrypted text, according to the dictionary. We can even be more creative and come up with better measures.

The content of the first few lines of the dictionary file dict.txt are the following.

```
aardvark
aardwolf
aaron
aback
abacus
abaft
abalone
abandon
abandoned
abandonment
abandons
abase
```

Note the dictionary file is sorted alphabetically. We can take advantage of this fact in our code.

Below is the main() function for our program.

```
//Do not change the main() function
int main(int argc, char *argv[])
{
    if(argc != 2)
    {
        printf("%s encrypted-message\n", argv[0]);
        return 0;
    }

    read_file_to_array("dict.txt");

    int encrypted[MAX];
    int len = read_encrypted(argv[1], encrypted);

    char message[MAX];
    strcpy(message, "");
    search(encrypted, len, message);
    printf("%s\n", message);
    return 0;
}
```

The function search() is already implemented as following.

```
//search using all the (key, shift) combinations
//to find the original message
//result is saved in message
void search(const int *encrypted, int len, char *message)
{
    char decrypted[MAX];

    int max_score = 0;
    strcpy(message, "");
    for(unsigned char k = 0; k < 255; k++)
    {
        for(unsigned char shift = 0; shift <= 24; shift++)
        {
            decryption(k, shift, encrypted, len, decrypted);
            int score = message_score(decrypted);
            if(score > max_score)
            {
                max_score = score;
                strcpy(message, decrypted);
            }
        }
    }
}
```

We need to implement the following functions.

```
//TODO
//Test wether a string word is in the dictionary
//Return 1 if word is in the dictionary
```

```
//Return 0 otherwise
//Be efficient in implementing this function
//Efficiency is needed to pass test cases in limited time
int in_dict(char *word)
{




}
//TODO
//Use key and shift to decrypt the encrypted message
//len is the length of the encrypted message
//Note the encrypted message is stored as an array of integers (not chars)
//The result is in decrypted

void decryption(unsigned char key, unsigned char shift, const int *encrypted, int len, char *dec
rypted)
{




}

//TODO
//calculate a score for a message msg
//the score is used to determine whether msg is the original message
int message_score(const char *msg)
{









}
//TODO
//read the encrypted message from the file to encrypted
//return number of bytes read
int read_encrypted(char *filename, int *encrypted)
{




}
```