```
//  Shane Stacy, CIS 241-03
////////////////////////////////////////////////////////////
//  c source code
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

static int num2 = 0;
static char c;
static FILE *input;
static FILE *output;

int found (char list[], int n, char ch);
void initializeFiles(char input1[], char output1[]);
void closeFiles();
void checkForSpace();

int main(int argc, char* argv[]) {

	//  int argc is 4
	//  argv[0] is the program
	//  argv[1] is the operation specifier (0 for encrypt, 1 for decrypt)
	//  argv[2] is the key
	//  argv[3] is the input file
	//  argv[4] is the output file
	//  example arguments:  ./proj1 0 key input.dat ouput.out

	char key[20], list[26];
	char alphabet[26] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z'};
	int i, d, size, num;
	size = 0;

	num = atoi(argv[1]);
	strcpy(key, argv[2]);
	printf("This is the key entered:  %s\n", key);
	printf("This is the default alphabet:  %s\n", alphabet);

	//  remove duplicate letters
	for (i = 0; i <= strlen(key); i++) {
		if(!found(list, size, key[i])) {
			list[size] = key[i];
			size++;
		}
	}

	printf("Duplicates removed from key:  %s\n", list);
```

```
size = strlen(list);

//  append the alphabet backwards
for (i = 0; i < strlen(alphabet) + 1; i++) {
        if(!found(list, size, alphabet[25 - i])) {
                list[size] = alphabet[25 - i];
                size++;
        }
}

printf("Encrypted alphabet:  %s\n", list);
d = strlen(list);
printf("Length of alphabet:  %d\n", d);

//  should the program encrypt or decrypt??
if (num == 0) {  //  decrypt
        printf("Decrypting...\n");

        initializeFiles(argv[3], argv[4]);

        //  while not end of file, decrypt
        while (!feof(input)) {
                c = getc(input);
                if (feof(input)) {
                        break;
                }

                checkForSpace();

                found(list, strlen(list), c);
                c = alphabet[num2];
                fprintf(output, "%c", c);
        }
        closeFiles();
        printf("FINISHED DECRYPTING\n");
}
else if (num == 1) {  //  encrypt
        printf("Encrypting...\n");

        initializeFiles(argv[3], argv[4]);

        //  while not end of file, encyrpt
        while (!feof(input)) {
                c = fgetc(input);
                if (feof(input)) {
                        break;
                }

                checkForSpace();
```

```
                found(alphabet, strlen(list), c);
                c = list[num2];
                fprintf(output, "%c", c);
            }
            closeFiles();
            printf("FINISHED ENCRYPTING\n");
        }
        else {  //  terminate the program if not 0 or 1
            printf("invalid encryption argument\n");
        }
        return 0;
}

//  helper method stores the index of a char if found in a string.  If found, return the index.  Else, return
0.
int found(char list[], int n, char target)  {
        int z;
        for (z=0; z<n; z++) {
            if (list[z] == target) {
            num2 = z;
            return 1;
            }
        }
        return 0;
}
//  defines the file streams
void initializeFiles(char input1[], char output1[]) {

        input = fopen(input1, "r");  //  open the input file
        output = fopen(output1, "w");  //  open the output file
}
//  closes file streams
void closeFiles() {

        fclose(input);
        fclose(output);
}

void checkForSpace() {

        if (isspace(c)) {
            fprintf(output, "\n");
            c = fgetc(input);
        }
}
//////////////////////////////////////////////////////////////////////
//  makefile
test: test1 test2 test3 test4 test5
```

```
test1:
	gcc proj1.c -o proj1

test2:
	./proj1 1 textbook encrypt.dat encryptOutput.out

test3:
	diff -s encryptOutput.out decrypt.dat

test4:
	./proj1 0 textbook decrypt.dat decryptOutput.out

test5:
	diff -s decryptOutput.out encrypt.dat
```

```
////////////////////////////////////////////////////////////////////
//  data to be encrypted
shane
eats
a
lot
of
cheeseburgers
////////////////////////////
//  data to be decrypted
jytqo
otij
t
spi
pk
xyoojoehlzolj
//////////////////////////////////
// encrypt output
jytqo
otij
t
spi
pk
xyoojoehlzolj
//////////////////////////////////
//  decrypt output
shane
eats
a
lot
of
cheeseburgers
```