```c
#include<stdio.h>
#include <string.h>
#include <stdlib.h>

#define NUMLETTERS 26
#define ASCIIA 97
#define ASCIIZ 122
#define SUCCESS 1
#define ERROR 0

struct enc_map_char{
   char pt_char;
   char ct_char;
};

struct enc_map_char enc_map[NUMLETTERS];

// Utility function: not given in the exam
/* The function initializes the encoding map array
*/
void fill_enc_map(){
   enc_map[0].pt_char = 'a'; enc_map[0].ct_char = 'q';
   enc_map[1].pt_char = 'c';enc_map[1].ct_char = 'a';
   enc_map[2].pt_char = 'd';enc_map[2].ct_char = 'p';
   enc_map[3].pt_char = 'b'; enc_map[3].ct_char = 'c';
   enc_map[4].pt_char = 'e';enc_map[4].ct_char = 'b';
   enc_map[5].pt_char = 'f';enc_map[5].ct_char = 'x';
   enc_map[6].pt_char = 'g';enc_map[6].ct_char = 'd';
   enc_map[7].pt_char = 'i';enc_map[7].ct_char = 'o';
   enc_map[8].pt_char = 'j';enc_map[8].ct_char = 'v';
   enc_map[9].pt_char = 'k';enc_map[9].ct_char = 'e';
   enc_map[10].pt_char = 'l';enc_map[10].ct_char = 'u';
   enc_map[11].pt_char = 'h';enc_map[11].ct_char = 'w';
   enc_map[12].pt_char = 'm';enc_map[12].ct_char = 'n';
   enc_map[13].pt_char = 'n';enc_map[13].ct_char = 'f';
   enc_map[14].pt_char = 'p';enc_map[14].ct_char = 'g';
   enc_map[15].pt_char = 'q';enc_map[15].ct_char = 'z';
   enc_map[16].pt_char = 'r';enc_map[16].ct_char = 'h';
   enc_map[17].pt_char = 'o';enc_map[17].ct_char = 't';
   enc_map[18].pt_char = 's';enc_map[18].ct_char = 'm';
   enc_map[19].pt_char = 't';enc_map[19].ct_char = 'i';
   enc_map[20].pt_char = 'u';enc_map[20].ct_char = 'l';
   enc_map[21].pt_char = 'w';enc_map[21].ct_char = 'y';
   enc_map[22].pt_char = 'x';enc_map[22].ct_char = 'k';
   enc_map[23].pt_char = 'y';enc_map[23].ct_char = 'r';
   enc_map[24].pt_char = 'v';enc_map[24].ct_char = 'j';
   enc_map[25].pt_char = 'z';enc_map[25].ct_char = 's';
}

// Utility function: not given in the exam
/* Function that checks if a character
*  is a lower-case letter of English Alphabet
*/
int (validate_ch)(char ch){
   if ((ch < ASCIIA) || (ch > ASCIIZ))
      return ERROR;
   else
      return SUCCESS;
}

// Utility function: not given in the exam
```

```c
/* Function that checks if a string
*  is composed of lower-case letters of English Alphabet
*/
int validate_pt(char * text){
    if (!text) return ERROR;
    if (!strlen(text)) return ERROR;

    for (int i = 0; i < strlen(text); i++)
        if (validate_ch(text[i]) == ERROR)
            return ERROR;

    return SUCCESS;
}

// Utility function: not given in the exam
/* Function for printing the encoding map
*/
int print_map(const struct enc_map_char * const enc_map){
    if (!enc_map) return ERROR;
    for (int i = 0; i < NUMLETTERS; i++)
        printf("%c %c\n", enc_map[i].pt_char, enc_map[i].ct_char);
    return SUCCESS;
}


/******************
* Question 1
******************/
char enc_shifting_char(char pt_char, int key){

    /* The function assumes that
    * -- pt_char is a lower-case letter of English alphabet
    * -- key is an integer.
    */

    // Bring the key into the [1-25] range
    key = key % NUMLETTERS;

    int ct_char = pt_char + key;

    // Bring cC into the valid range of ASCII codes
    if (ct_char > ASCIIZ)
        ct_char = ASCIIA - 1 + ct_char % ASCIIZ;
    else if (ct_char < ASCIIA)
        ct_char = ASCIIZ + 1 - (ASCIIA - ct_char) ;

    return ct_char;
}

/******************
* Question 2
******************/
char* enc_shifting_string(char *pt, char *ct, int key){

    // Check the validity of the input parameters
    // - pt and ct must not be NULL pointers
    // Function accepts any key
    if (pt == NULL || ct == NULL)
        return NULL;

    char* p = ct;
```

```
    for (int i = 0; i < strlen(pt); i++){
        char ct_char = enc_shifting_char(pt[i], key);
        if (validate_ch(ct_char) == ERROR) return NULL;
        *p++ = ct_char;
    }

    // Remember to insert NULL char at the end of the string
    *p = '\0';
    return ct;
}

/******************
* Utility function, not given in the exam
******************/
int validate_enc_map(const struct enc_map_char * enc_map){

    /* Check the validity of enc_map_char array.
    * The array is valid if:
    * - in all (plaintext, ciphertext) pairs,
    *     plaintext char is *different* from the ciphertext char
    * - all plaintext chars are unique
    * - all ciphertext chars are unique
    * - all plaintext and ciphertext chars are lower-case English Alphabet characters
    */
    if (!enc_map) return ERROR;

    for (int i = 0; i < NUMLETTERS; i++){
        if (enc_map[i].pt_char == enc_map[i].ct_char) return ERROR;
        if (validate_ch(enc_map[i].pt_char) == ERROR) return ERROR;
        if (validate_ch(enc_map[i].ct_char) == ERROR) return ERROR;
        for (int j = i + 1; j < NUMLETTERS; j++){
            if (enc_map[i].pt_char == enc_map[j].pt_char) return ERROR;
            if (enc_map[i].ct_char == enc_map[j].ct_char) return ERROR;
        }
    }

    return SUCCESS;
}

/******************
* Question 3
******************/
int sort_enc_map(struct enc_map_char * enc_map){

    if (!enc_map) return ERROR;

    /* Selection sort
    */
    for (int i = 0; i < NUMLETTERS - 1; i++) {
        // Find the index *min* of the plaintext char
        // with the smallest ASCII code
        // in the range of indices [i+1, NUMLETTERS-1]
        int index_min = i;
        for (int j = i + 1; j < NUMLETTERS; j++)
            if (enc_map[j].pt_char < enc_map[index_min].pt_char)
                index_min = j;

        // If the index found is different than *i*
        // -- swap the elements at index *min* and *i*
        if (i != index_min) { // This check is not mandatory
```

```
            struct enc_map_char temp = {enc_map[index_min].pt_char, enc_map[index_min].ct_char};
            enc_map[index_min].ct_char =  enc_map[i].ct_char;
            enc_map[index_min].pt_char =  enc_map[i].pt_char;
            enc_map[i].pt_char =  temp.pt_char;
            enc_map[i].ct_char =  temp.ct_char;
        }
    }

    return SUCCESS;
}


/******************
* Question 4
******************/
char* enc_permutation_string(char * pt, char * ct, const struct enc_map_char * enc_map){

    if (!pt || !ct || !enc_map) return NULL;

    char* p = ct;
    for (int i = 0; i < strlen(pt); i++)
        *p++ = enc_map[pt[i] - ASCIIA].ct_char;

    return ct;
}

/******************
* Question 5
******************/
int main(int argc, char *argv[]){

    // Check the input parameters
    if (argc < 2){
        printf("Too few input parameters.\n");
        return 0;
    }

    char* pt = argv[1];
    int key = 0;
    if (argc == 3) key = atoi(argv[2]);

    // Check if the input string is valid
    if (validate_pt(pt) == ERROR) {
        printf("Invalid input string.\n");
        return 0;
    }

    // Dynamically allocate the memory for the output string
    char* ct = calloc(strlen(pt) + 1, sizeof(char));
    if (ct == NULL) {
        printf("Encryption failed.\n");
        return 0;
    }

    if (argc == 3) { // Sliding encryption scheme
        // Check that key is not a multiple of NUMLETTERS
        if (key % NUMLETTERS == 0) {
            printf("Invalid key.\n");
            return 0;
        }
        ct = enc_shifting_string(pt, ct, key);
```

```
    }
    else { // Permutation encryption scheme
        // Fill, validate, and sort the encoding map
        fill_enc_map();
        sort_enc_map(enc_map);
        if (validate_enc_map(enc_map) == ERROR){
            printf("Invalid encoding map.\n");
            return 0;
        }
        ct = enc_permutation_string(pt, ct, enc_map);
    }

    if (ct)
        printf("%s\n", ct);
    else
        printf("Encryption failed.\n");

    // Free the allocated memory
    free(ct);
    ct = NULL;

    return 0;
}
```