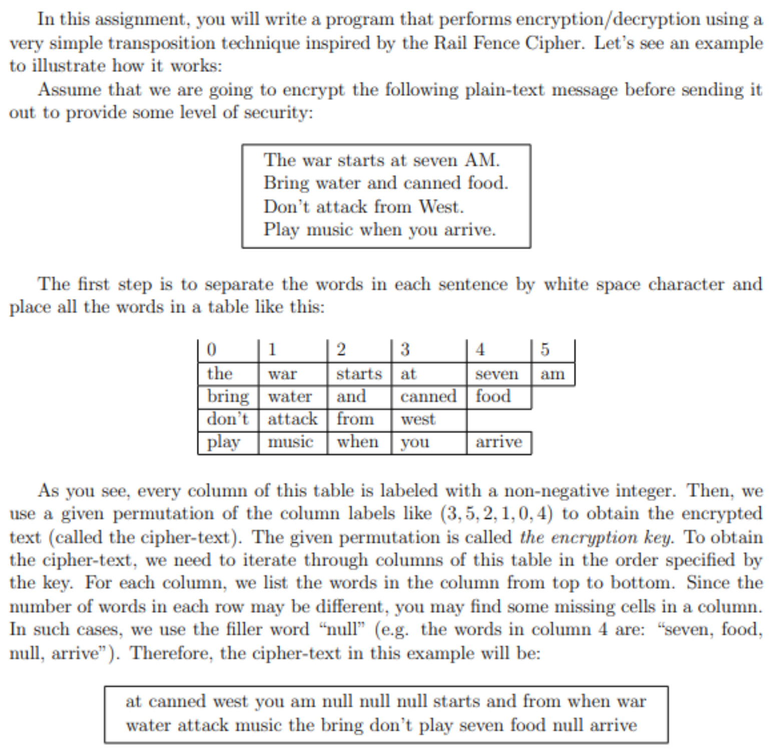
Question: In this assignment, you will write a program that performs enc…

Need to create the decryption file given this code.



In this assignment, you will write a program that performs encryption/decryption using a very simple transposition technique inspired by the Rail Fence Cipher. Let's see an example to illustrate how it works:

Assume that we are going to encrypt the following plain-text message before sending it out to provide some level of security:

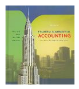
```
The war starts at seven AM.
Bring water and canned food.
Don't attack from West.
Play music when you arrive.
```

The first step is to separate the words in each sentence by white space character and place all the words in a table like this:

0	1	2	3	4	5
the	war	starts	at	seven	am
bring	water	and	canned	food	
don't	attack	from	west		
play	music	when	you	arrive	

As you see, every column of this table is labeled with a non-negative integer. Then, we use a given permutation of the column labels like $(3, 5, 2, 1, 0, 4)$ to obtain the encrypted text (called the cipher-text). The given permutation is called *the encryption key*. To obtain the cipher-text, we need to iterate through columns of this table in the order specified by the key. For each column, we list the words in the column from top to bottom. Since the number of words in each row may be different, you may find some missing cells in a column. In such cases, we use the filler word "null" (e.g. the words in column 4 are: "seven, food, null, arrive"). Therefore, the cipher-text in this example will be:

```
at canned west you am null null null starts and from when war
water attack music the bring don't play seven food null arrive
```



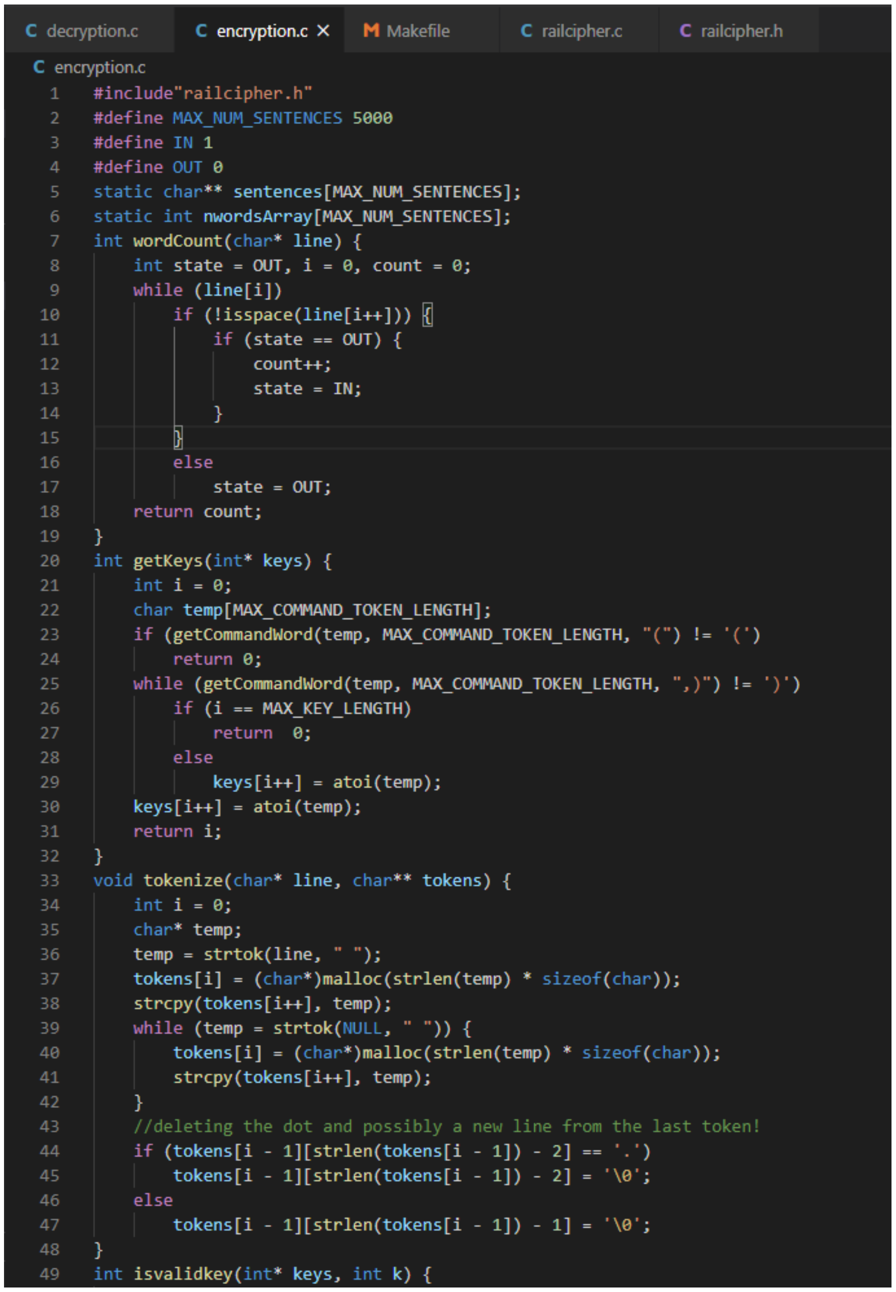
The recipient of this message can decrypt the above cipher-text easily as long as they know what the key is (in this example, $(3, 5, 2, 1, 0, 4)$). Here are the steps for decryption and obtaining plain-text from cipher-text:

1. Count the number words n in the cipher-text (24 words in this example).
2. Divide the number of words ($n = 24$) by the length of given key ($k = 6$): $n/k = 4$.
3. Draw a table with n/k rows and k columns
4. Place words in the columns of drawn table using the order given by the key.
5. Extract each sentence from each row of the table after removing the filler (null) words.

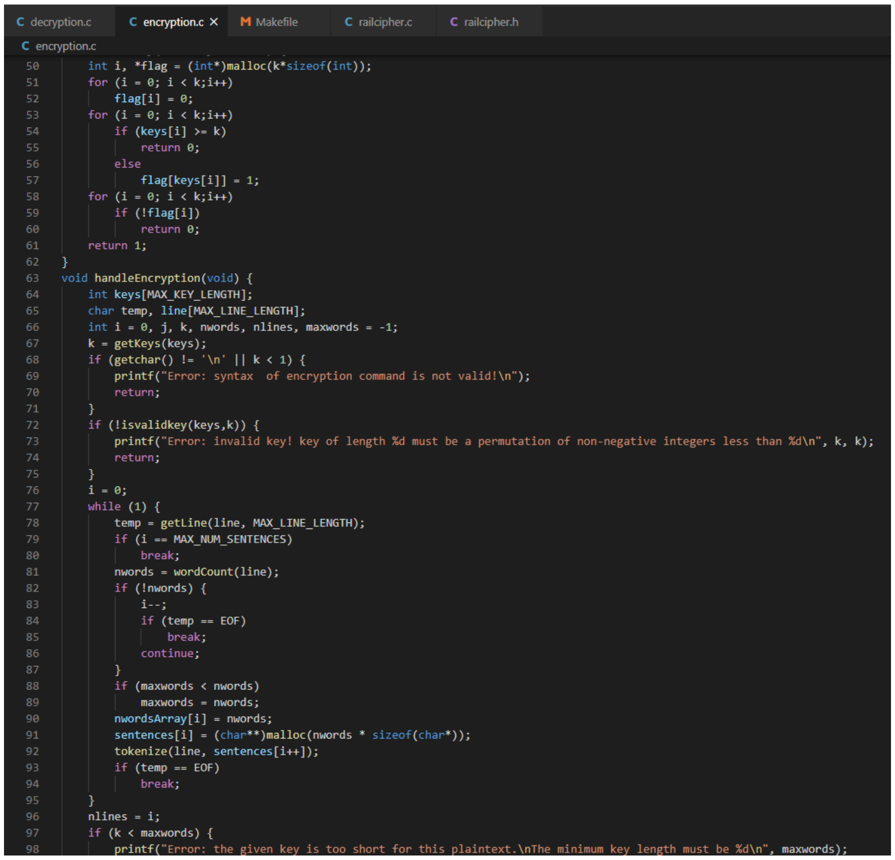
1 Program Input Commands

There are three valid commands for your program:

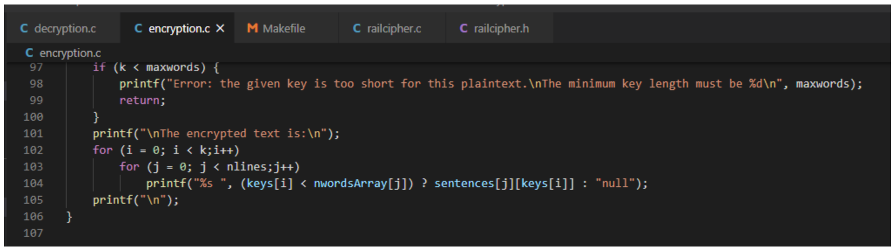
- encrypt $(a_0, a_1, \ldots, a_{k-1})$ ↵ PLAINTEXT Ctrl D: encrypts the given plain-text using the given key which is a permutation of the first k non-negative integers. The plain-text starts with a new line character and ends with an EOF character which is entered via keyboard using Ctrl+D. This command prints the encrypted text on the screen. For the sake of simplicity, assume that:
  - Plain-text is made or one or more sentences (at-most 5000 sentences).
  - Every sentence ends with a dot (.).
  - Two consecutive sentences are separated by a single new line character.
  - Each sentence is made of at-most 100 words separated by white-space.
  - Each word in a sentence is made of only lowercase alphabetical letters or an apostrophe. Length of each word is at-most 24 characters.
  - No punctuation marks are used in the plain-text except apostrophe and dot.
- decrypt $(a_0, a_1, \ldots, a_{k-1})$ ↵ CIPHERTEXT Ctrl D: decrypts the given cipher-text using the given key which is a permutation of the first k non-negative integers. The cipher-text starts with a new line character and ends with an EOF character which is entered via keyboard using Ctrl+D. This command prints the decrypted text on the screen. For the sake of simplicity, assume that the cipher-text contains no new line character.
- quit: ends the program.



```c
#include"railcipher.h"
#define MAX_NUM_SENTENCES 5000
#define IN 1
#define OUT 0
static char** sentences[MAX_NUM_SENTENCES];
static int nwordsArray[MAX_NUM_SENTENCES];
int wordCount(char* line) {
    int state = OUT, i = 0, count = 0;
    while (line[i])
        if (!isspace(line[i++])) {
            if (state == OUT) {
                count++;
                state = IN;
            }
        }
        else
            state = OUT;
    return count;
}
int getKeys(int* keys) {
    int i = 0;
    char temp[MAX_COMMAND_TOKEN_LENGTH];
    if (getCommandWord(temp, MAX_COMMAND_TOKEN_LENGTH, "(") != '(')
        return 0;
    while (getCommandWord(temp, MAX_COMMAND_TOKEN_LENGTH, ",)") != ')')
        if (i == MAX_KEY_LENGTH)
            return  0;
        else
            keys[i++] = atoi(temp);
    keys[i++] = atoi(temp);
    return i;
}
void tokenize(char* line, char** tokens) {
    int i = 0;
    char* temp;
    temp = strtok(line, " ");
    tokens[i] = (char*)malloc(strlen(temp) * sizeof(char));
    strcpy(tokens[i++], temp);
    while (temp = strtok(NULL, " ")) {
        tokens[i] = (char*)malloc(strlen(temp) * sizeof(char));
        strcpy(tokens[i++], temp);
    }
    //deleting the dot and possibly a new line from the last token!
    if (tokens[i - 1][strlen(tokens[i - 1]) - 2] == '.')
        tokens[i - 1][strlen(tokens[i - 1]) - 2] = '\0';
    else
        tokens[i - 1][strlen(tokens[i - 1]) - 1] = '\0';
}
int isvalidkey(int* keys, int k) {
```

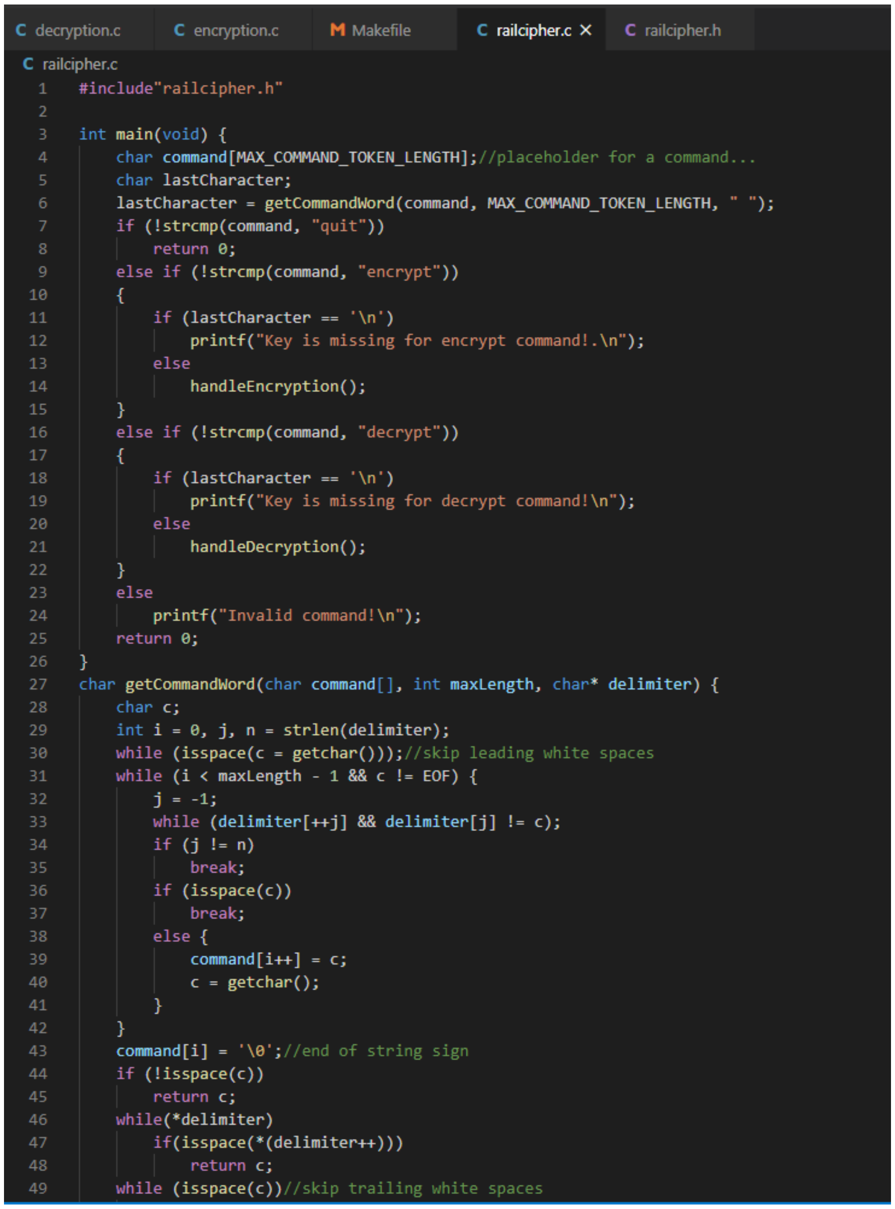


```c
    int i, *flag = (int*)malloc(k*sizeof(int));
    for (i = 0; i < k;i++)
        flag[i] = 0;
    for (i = 0; i < k;i++)
        if (keys[i] >= k)
            return 0;
        else
            flag[keys[i]] = 1;
    for (i = 0; i < k;i++)
        if (!flag[i])
            return 0;
    return 1;
}
void handleEncryption(void) {
    int keys[MAX_KEY_LENGTH];
    char temp, line[MAX_LINE_LENGTH];
    int i = 0, j, k, nwords, nlines, maxwords = -1;
    k = getKeys(keys);
    if (getchar() != '\n' || k < 1) {
        printf("Error: syntax  of encryption command is not valid!\n");
        return;
    }
    if (!isvalidkey(keys,k)) {
        printf("Error: invalid key! key of length %d must be a permutation of non-negative integers less than %d\n", k, k);
        return;
    }
    i = 0;
    while (1) {
        temp = getLine(line, MAX_LINE_LENGTH);
        if (i == MAX_NUM_SENTENCES)
            break;
        nwords = wordCount(line);
        if (!nwords) {
            i--;
            if (temp == EOF)
                break;
            continue;
        }
        if (maxwords < nwords)
            maxwords = nwords;
        nwordsArray[i] = nwords;
        sentences[i] = (char**)malloc(nwords * sizeof(char*));
        tokenize(line, sentences[i++]);
        if (temp == EOF)
            break;
    }
    nlines = i;
    if (k < maxwords) {
        printf("Error: the given key is too short for this plaintext.\nThe minimum key length must be %d\n", maxwords);
```



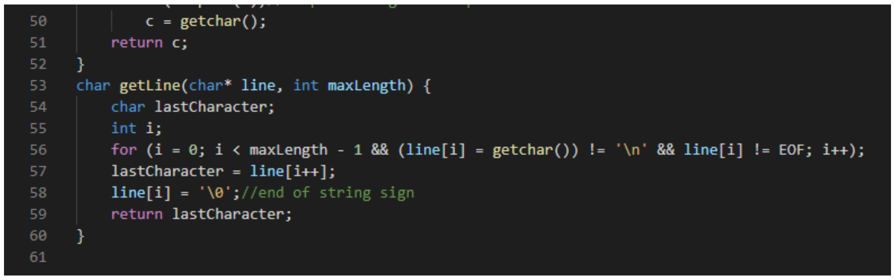
```c
    if (k < maxwords) {
        printf("Error: the given key is too short for this plaintext.\nThe minimum key length must be %d\n", maxwords);
        return;
    }
    printf("\nThe encrypted text is:\n");
    for (i = 0; i < k;i++)
        for (j = 0; j < nlines;j++)
            printf("%s ", (keys[i] < nwordsArray[j]) ? sentences[j][keys[i]] : "null");
    printf("\n");
}

```



```c
#include"railcipher.h"

int main(void) {
    char command[MAX_COMMAND_TOKEN_LENGTH];//placeholder for a command...
    char lastCharacter;
    lastCharacter = getCommandWord(command, MAX_COMMAND_TOKEN_LENGTH, " ");
    if (!strcmp(command, "quit"))
        return 0;
    else if (!strcmp(command, "encrypt"))
    {
        if (lastCharacter == '\n')
            printf("Key is missing for encrypt command!.\n");
        else
            handleEncryption();
    }
    else if (!strcmp(command, "decrypt"))
    {
        if (lastCharacter == '\n')
            printf("Key is missing for decrypt command!\n");
        else
            handleDecryption();
    }
    else
        printf("Invalid command!\n");
    return 0;
}
char getCommandWord(char command[], int maxLength, char* delimiter) {
    char c;
    int i = 0, j, n = strlen(delimiter);
    while (isspace(c = getchar()));//skip leading white spaces
    while (i < maxLength - 1 && c != EOF) {
        j = -1;
        while (delimiter[++j] && delimiter[j] != c);
        if (j != n)
            break;
        if (isspace(c))
            break;
        else {
            command[i++] = c;
            c = getchar();
        }
    }
    command[i] = '\0';//end of string sign
    if (!isspace(c))
        return c;
    while(*delimiter)
        if(isspace(*(delimiter++)))
            return c;
    while (isspace(c))//skip trailing white spaces
```



```c
        c = getchar();
    return c;
}
char getLine(char* line, int maxLength) {
    char lastCharacter;
    int i;
    for (i = 0; i < maxLength - 1 && (line[i] = getchar()) != '\n' && line[i] != EOF; i++);
    lastCharacter = line[i++];
    line[i] = '\0';//end of string sign
    return lastCharacter;
}
```

Show transcribed image text

Expert Answer

Anonymous answered this
92 answers

Was this answer helpful? 1 0

Answer:

Here is the code for decryption:

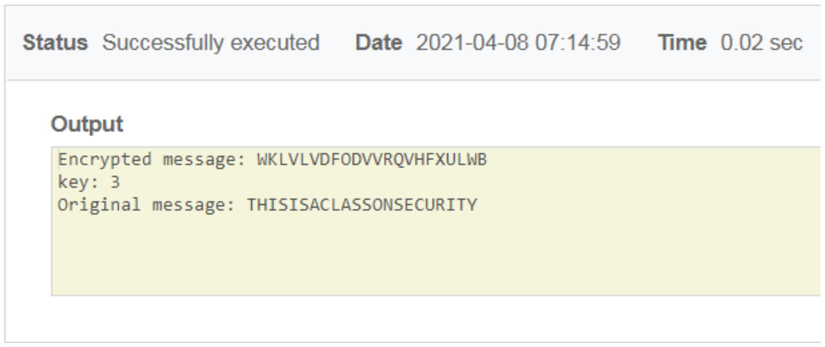
```
def decrypt(text,s):
    result = ""

    for i in range(len(text)):
        char = text[i]
        result += chr((ord(char)-s-65)%26+65)

    return result

cipher = "WKLVLVDFODVVRQVHFXULWB"
s = 3
print("Encrypted message: " + cipher)
print("Key: " + str(s))
print("Original message: " + decrypt(cipher,s))
```

Here is the output:



Status Successfully executed Date 2021-04-08 07:14:59 Time 0.02 sec

Output

```
Encrypted message: WKLVLVDFODVVRQVHFXULWB
key: 3
Original message: THISISACLASSONSECURITY
```

We can see that the decryption function works perfectly fine as the original message that is printed is correct.

i. Drawbacks of Caesar cipher: The Caesar cipher is a very weak cipher, it can easily be broken as the key can be found quiet easily using all the combinations. So this cipher has very weak security.

ii. For improving Caesar technique: We can substitute 2 values for 1 character, one with (char + key) and other with (char - key), or we can use some difficult formula for each encryption of each character.

Hope you like the solution :)