```c
/*
*********************************************************************************
*                                CEASAR
*********************************************************************************

Command format:

              ceasar <input.txt> <output.txt> <key> <row>

It encodes the text in <input.txt> with Ceasar coding. The coding key is a
positive integer <key>. It encodes a sentence indicated by <row> (a positive
integer), between 2 consecutive full stop characters.
The encoded sentence is printed in <output.txt> (as a verification also the
encoded alphabet is printed).
The text in <input.txt> is saved into a 2D array where each row contains one
sentence (including the full stop character).
The correctness of the command format is checked.
*********************************************************************************
*/

# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include <ctype.h>

# define MAX 1000

/* To abbreviate */
typedef short unsigned int S_U_INT;

/* Function Prototypes */
unsigned int FtoM(FILE*, char[][MAX]);
void CeasarCoding(char[], S_U_INT, char[]);



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

    /*
    argv+1 contains the name of the input file.
    argv+2 contains the name of the input file.
    argv+3 contains the key value.
    argv+4 contains the row to encode.
    */

    /* Check number of arguments */
    if (argc != 5) {
        printf("\n Incorrect Command!\n");
        return 0;
    }

    FILE* fin;
    unsigned int rows, sentence;
    char m[MAX][MAX];

    fin  = fopen(*(argv + 1), "r");
    rows = FtoM(fin, m);
    sentence   = atoi(*(argv + 4));
```

```c

    /* Check the value of <row> */
    if (sentence > rows) {
        printf("\nThe sentence numbers is higher than the ");
        printf("number of sentences %s", *(argv + 1));
        return 0;
    }

    sentence--;    /* READ BELOW */

    char out[100];
    S_U_INT key;

    key = atoi(*(argv + 3));
    strcpy(out, *(argv + 2));
    CeasarCoding(m[sentence], key, out);
    fclose(fin);
}

/* The value of 'sentence' must be decremented by 1 because the rows of 'm' have
 indeces starting from 0 */

 /*****************************************************************************
 *****************************************************************************



 *****************************************************************************
 *                           CeasarCoding
 *****************************************************************************
 Receives a string 's' as input, an unsigned short integer 'key' and a string
 'out' containing the name of the output file.
 It performs the Ceasar coding using the given key.

 It checks the value of 'key' (must be smaller than 26). It also checks that
 's' actually contains text.

 Two strings 'a' and 'A' to contain the letters of the English alphabet (the
 length of these strings is 26 plus one, accounting for the end of string
 character '/0') and to treat normal and capital letters.
 The coding alphabets are built into 'ak' and 'AK' from 'a' and 'A' using
 'key': The first k letters in these alphabets are the last k in the normal
 alphabet.

 The indeces to address the letterns in the alphabets can be calculated as

 s[i] - 'a' (for non-capital letters).

 The calculated index can be used in the coded alphabet to get the coded
 character.

 NB The function doesn't take into account special characters.
 */



```

```c

void CeasarCoding(char s[], S_U_INT key, char out[]) {
    /* Check value of 'key' */
    if (key > 25)
        printf("\nKey must be smaller than 26!\n");

    /* Check that 's' contains text */
    else if ((s[0] == '.' && s[1] == '\0') || s[0] == '\0') {
        printf("\nNo text!\n");
    }
    else {
        FILE * fout;

        short int j;
        char ak[27], AK[27];

        char a[] = "abcdefghijklmnopqrstuvwxyz";
        char A[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        fout = fopen(out, "w");

        j = 26 - key;

        strcpy(ak, (a + j));
        strcpy(AK, (A + j));
        strncat(ak, a, j);
        strncat(AK, A, j);

        for (int i = 0; s[i] != '\0'; i++) {
            if (s[i] >= 'a' && s[i] <= 'z') {
                j = s[i] - 'a';
                putc(ak[j], fout);
            }
            else if (s[i] >= 'A' && s[i] <= 'Z') {
                j = s[i] - 'A';
                putc(AK[j], fout);
            }
            else
                putc(s[i], fout);
        }

        fprintf(fout, "\n\n%s\n%s\n", A, AK);
        fprintf(fout, "\n%s\n%s\n\nEND.", a, ak);
        printf("\nRead the result in %s\n", out);

        fclose(fout);
    }
}


/*****************************************************************************
******************************************************************************
```

```c
/*
*****************************************************************************
*                               FtoM
*****************************************************************************

Receives a pointer to a file and a 2D array of char.
Copies each sentence in a row of the array and returns the number of rows that
have been filled, which is the number of sentences.

It ignores empty spaces at the beginning.

To determine the end of a sentence it checks that an empty space, a new line or
a tabulation character follows a full stop character (the latter is copied too).
A new line character in the text is translated into an empty space in the array.
If other characters follow a full stop character, it continues to copy in the
same sentence.

BEWARE: other types of mistakes (e.g. punctuation characters after '.') are
not checked.
*/


unsigned int FtoM(FILE* f, char m[][MAX]) {
    int c, i = 0, j = 0;

    while ((c = getc(f)) != EOF && j < MAX) {
        /* Initial empty spaces */
        if (isspace(c) && j == 0)
            continue;
        /* Copy everything that isn't '.' or '\n'*/
        else if (c != '.' && c != '\n')
            m[i][j++] = c;
        else if (c == '.') {
            m[i][j++] = c;
            c = getc(f);
            if (isspace(c)) {
                m[i++][j] = '\0';
                j = 0;
            }
            else if (c == EOF) {
                m[i][j] = '\0';
                break;
            }
            else
                m[i][j++] = c;
        }
        else
            m[i][j++] = ' ';
    }
    return i + 1;
}
```