```c
/*****************************************************************************
*                                  ecoCond2
*****************************************************************************

                  ecoCond2  <input.txt>  <mobile>  <ref>  <lim>

Reads a text file and copies phrases (delimited by full stops) on another text files if
these phrases contains a number of occurrences of <mobile> preceeding an instance of <ref>
higher than <lim>.

NB: the name of the output file is fixed to be 'filteredOcc.txt'.
*/


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

# define MAX 1000

int FtoM(FILE *, char [][MAX]);
int occurrence(char [], char [], char []);

int main (int argc, char * argv[]) {
    if (argc!= 5) {
        printf("\nInvalid Command!\n");
        return 0;
    }

    if (*(argv + 4)[0] == '-') {
        printf("\nError: negative numbers not allowed!\n");
        return 0;
    }

    unsigned short lim;
    lim = atoi (*(argv + 4));

    FILE *fin, *fout;
    char m[MAX][MAX];
    int rows, instances, counter=0;

    fin  = fopen (*(argv + 1), "r");
    fout = fopen ("filteredOcc.txt", "w");

    rows = FtoM(fin, m);

    if (lim == 1)
        fprintf(fout, "Phrases with more than 1 instance of <%s> before <%s>:\n\n", *(argv
           + 2), *(argv + 3));
    else
        fprintf(fout, "Phrases with more than %d instance of <%s> before <%s>:\n\n", lim,
          *(argv + 2), *(argv + 3));

    for (int i = 0; i < rows; i++) {
        if ( (instances = occurrence(m[i], *(argv + 2), *(argv + 3)) ) > lim) {
            counter++;
            fprintf(fout, "Phrase n. %d:\n%s (Instances: %d) \n\n", i + 1, m[i],
```

```
                instances);
57          }
58      }
59
60      if (counter == 0)
61          fprintf (fout, "None.\n\n");
62
63      printf ("\nRead results in filteredOcc.txt\n");
64
65      fclose(fin);
66      fclose(fout);
67      return 0;
68  }
69
70
71
72
73
74  /*****************************************************************************
75  *                                    occurrence
76  *****************************************************************************
77
78  Receives 3 strings, 's', 'mobile', and 'ref'. 's' must be terminated by full stop. The
79  function counts the number of occurrences of 'mobile' before an instance of 'ref'
80  contained in 's'.
81
82  Any punctuation is ignored except '.' which is either the end of the string or in between
83  words (in which case the 2 words are considered as 1).
84
85  An enumeration, 'state', is used to detect whether we are into a word or not. When
      state=in,
86  if an empty space or a punctuation is read, a word is considered concluded. When
      state=out,
87  empty spaces and punctuation are ignored; alphanumerical characters cause a change of
      'state'
88  to 'in' and the word starts to be saved.
89
90  A string buffer 'word' is used to store words as they're read. If the word saved is equal
91  to 'mobile', a count is incremented; if the word saved is equal to 'ref', the count is
92  returned.
93  */
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
```

```c


int occurrence(char s[], char mobile[], char ref[]) {
    if (strcmp(mobile, ref) == 0)
        return 0;

    int j = 0, count = 0;
    enum state { out , in } state;
    char word[31];

    state = in;

    for (int i = 0; s[i] != '\0'; i++) {
        if (state == in) {
            if ( isspace(s[i]) || ispunct(s[i]) && s[i] != '.') {
                state = out;
                word[j] = '\0';
                j = 0;

                if(strcmp(word , mobile) == 0)
                    count++;
                else if (strcmp(word , ref) == 0)
                    return count;
            }
            else if ( s[i] == '.' && s[i + 1] == '\0' ) {
                word[j] = '\0';
                break;
            }
            else if ( s[i] == '.' && isalnum(s[i + 1]) )
                word[j++] = s[i];
            /* Alphanumerical characters */
            else
                word[j++] = s[i];
        }
        /* state == out */
        else {
            if ( isspace(s[i]) || ispunct(s[i]) )
                continue;
            /* Alphanumerical characters */
            else {
                state = in;
                word[j++] = s[i];
            }
        }
    }

    /* Here the end of the string is reached and if the last word is equal to 'ref', the
       count is returned; otherwise 'ref' isn't found and 0 is returned. */
    if (state == in && strcmp(word, ref) == 0)
        return count;
    else
        return 0;

}
```

```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.
*/

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

    while ((c = getc(f)) != EOF && j < MAX) {
        if (isspace(c) && j == 0)
            continue;
        else if (c != '.' && c != '\n')
            m[i][j++] = c;
        else if (c == '.') {
            m[i][j++] = c;

            if (isspace((c = getc(f)))) {
                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;
}
```