

**”TEXT EDITOR LIBRARY”**

**TĂNĂSOIU TEODORA**

**CR3 S1B**

## 1 Problem statement

Write a library with functions that could be useful for a text editor. The library should at least contain the following functions:

- determine if a word is present in a document
- number of occurrences for a word in a document
- replace all instances of a word in a document with another given word
- replace specific instances of a word in a document with another given word
- create a sorted list of words by the number of occurrences in a document
- delete consonants

The library is useful as a text editor for the user. The input is read from a text file and the design offers the opportunity to choose the function the user needs for editing the text he desires.

## 2 Pseudocode

### **FIND(s, word)**

1. Open file "in.txt"
2. **if** f == NULL **then**
3.   ▷ "Missing or empty file!"
4. **while**(reading s until EOF)
5.     **if**(**strcmp**(s, word) == 0)*then*
6.       **return** 1
7. **return** 0
8.     Close file "in.txt"

### **OCCURRENCE(s, word)**

1. *counter* ← 0
2. Open file "in.txt"
3. **if** f == NULL **then**
4.   ▷ "Missing or empty file!"
5. **while**(reading s until EOF)
6.     **if**(**strcmp**(s, word) == 0)*then*
7.       *counter*++
8. **return** *counter*
9.     Close file "in.txt"

### **REPLACE(to be replaced, replacement)**

1. Open files "in.txt", "tmp.txt"
2. **if** Input == NULL **then**
3.   ▷ "Empty or missing file!"

```

4.   return 1
5. while fgets(Buffer, 2000, Input) ≠ NULL do
6.   Stop=NULL
7.   Start=Buffer
8. while (1) do
9.   Stop ← strstr(Start, to be replaced)
10.  if Stop==NULL do
11.    fwrite(Start, 1, strlen(Start), Output)
12.    break()
13.  fwrite(Start, 1, Stop-Start, Output)
14.  fwrite(replacement, 1, strlen(replacement), Output)
15.  Start=Stop+strlen(to be replaced)
16. close files "in.txt", "out.txt"
17. remove "in.txt"
18. rename (TemporaryFileName, "in.txt")

```

### 3 Application design

\*The source code for the functions is developed using C language.

#### \*specification of the input:

The input consists of a file in which the text to be managed by the program is "stored"("in.txt").The text represents strings, "char" type variables.

#### \*specification of the output:

The result is printed on the screen as a result of compiling the program for the functions that contain the "printf message" and for the functions that update the input text file, the result will be printed in a temporary file ("tmp.txt").

#### \*list of the modules, functions and their description

- **main.c**, where we find:

- the declaration and the initialisations of the variables
- the call for the functions in the library
- the opening structure of the input file("in.txt")

-**functions:**(each function has its own ".c" file)

- find.c (s, word)
  - determines if a given word, read from the keyboard (word) exists in a string(s) read from a text file ("in.txt")

-it returns 1 if the word exists in the input text and it shows to the user the following message: "Yes, the word exists in this document!"

- occurrence.c (s, word)
  - determines the number of the occurrences of a given word, read from the keyboard (word) after it verifies its existence
  - it returns a counter (counter) and shows to the user the following message: "This word appears in the document for counter's value time[s]!"
- replace.c (to be replaced, replacement)
  - replaces all instances of a given word from the text (to be replaced) in the string (s) with another given word, chosen by the user, read from the keyboard (replacement) after it verifies its existence using "find"
  - returns a new, updated string, printed in a temporary file, "tmp.txt"
- replace specific instances.c (to be replaced, replacement, position)
  - using "find" and "occurrence", it verifies if the word exists in the file and counts its occurrences in order to make the replacement of the given word (to be replaced) with its substitute (replacement)
  - returns a new, updated string printed in a temporary file, "tmp.txt"
- sorted list.c
  - creates a sorted list of the words by the number of their occurrences in the document (using a structure: an int type for the number of occurrences and a char type for the words in the file)
  - using "occurrence" it determines the number of occurrences of every word in the text and after this counting, it makes an ascending sorting using a version of bubble sort
  - the result is printed on the screen consisting in a message with the number of occurrences for each word
- delete consonants.c
  - it deletes consonants using a "switch case" structure
  - it prints on the screen the first line of the text after removing all the consonants, actually showing only the vowels in the first line

- **headers:**

- find.h
- occurrence.h
- replace.h
- replace specific instances.h

- sorted list.h
- delete consonants.h

## 4 Conclusions

As a conclusion, I can say that the project itself, as a concept, was very challenging especially because it was something new, unexperienced before. I managed to finish all the tasks concerning the source code, but it definitely needs improvements in order to really help someone to edit his text. For example, I considered the input as a text without any punctuation marks or any other kind of symbols, which, normally, cannot be ignored. Also, the "create list" prints the message with the number of occurrences for several times (the number of occurrences) when it should only print it once.

## 5 References

- 1) "C++, Teorie si aplicatii", ed. "else".
- 2) 10-th grade problem textbook.
- 3) "Aplicatii in C", ed. DP.

## 6 Experiments and results

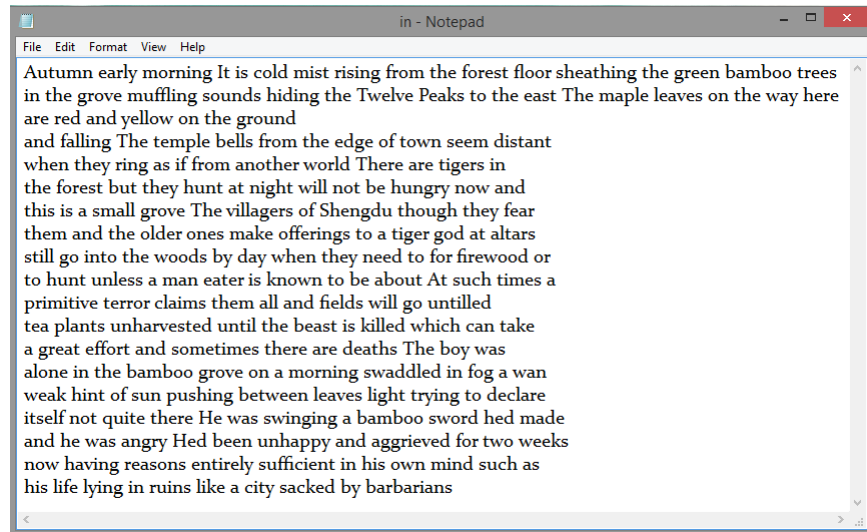


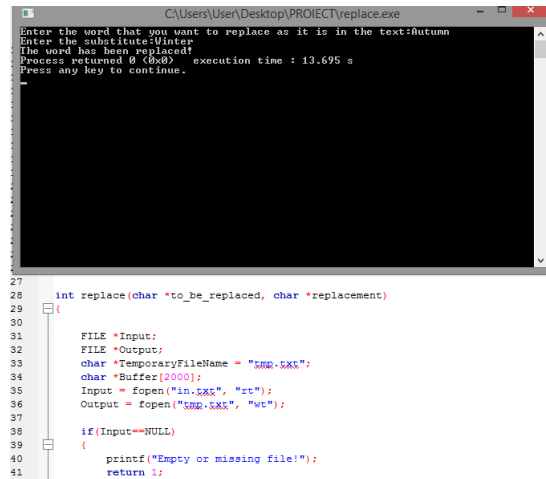
Figure 1: Input

```
21
22 int find (char *s, char *word)
23 {
24     f=fopen("in.txt","rt");
25     if (f == '0')
26     {
27         printf("Empty or missing file!");
28         getch();
29     }
30     else
31     {
32         while(fscanf(f,"%s",s)>=0)
33         {
34             if (strcmp(word,s)==NULL)
35             {
36                 return 1;
37             }
38         }
39     }
40 }
```

Figure 2: find

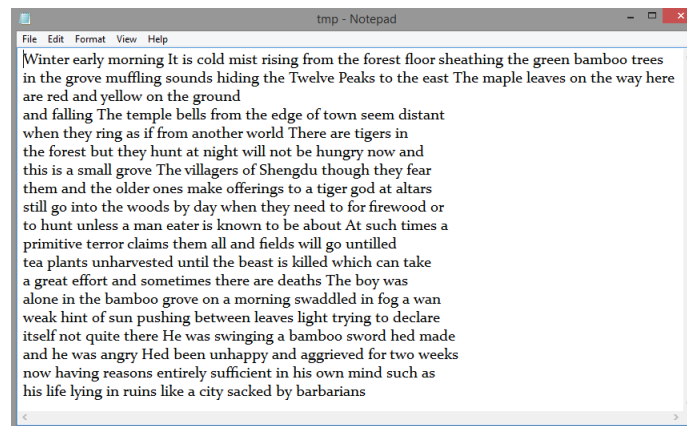
```
25 int occurrence(char *s, char *word)
26 {
27     int counter=0;
28     f=fopen("in.txt","rt");
29     while(fscanf(f,"%s",s)>=0)
30     {
31         if(strcmp(s,word)==NULL)
32         {
33             counter++;
34         }
35     }
36     return counter;
37 }
38 fclose(f);
39 getch();
40
41
```

Figure 3: occurrence



```
27
28 int replace(char *to_be_replaced, char *replacement)
29 {
30
31     FILE *Input;
32     FILE *Output;
33     char *TemporaryFileName = "tmp.txt";
34     char *Buffer[2000];
35     Input = fopen("in.txt", "rt");
36     Output = fopen(TemporaryFileName, "wt");
37
38     if(Input==NULL)
39     {
40         printf("Empty or missing file!");
41         return 1;
```

Figure 4: replace



Winter early morning It is cold mist rising from the forest floor sheathing the green bamboo trees  
in the grove muffling sounds hiding the Twelve Peaks to the east The maple leaves on the way here  
are red and yellow on the ground  
and falling The temple bells from the edge of town seem distant  
when they ring as if from another world There are tigers in  
the forest but they hunt at night will not be hungry now and  
this is a small grove The villagers of Shengdu though they fear  
them and the older ones make offerings to a tiger god at altars  
still go into the woods by day when they need to for firewood or  
to hunt unless a man eater is known to be about At such times a  
primitive terror claims them all and fields will go untilled  
tea plants unharvested until the beast is killed which can take  
a great effort and sometimes there are deaths The boy was  
alone in the bamboo grove on a morning swaddled in fog a wan  
weak hint of sun pushing between leaves light trying to declare  
itself not quite there He was swinging a bamboo sword hed made  
and he was angry Hed been unhappy and aggrieved for two weeks  
now having reasons entirely sufficient in his own mind such as  
his life lying in ruins like a city sacked by barbarians

Figure 5: replace

```

57
58 int specific_replacement(char *to_be_replaced, char *replacement, int position)
59 {
60
61     FILE *Input,*Output;
62     char Buffer[2000];
63     char *TemporaryFileName = "tmp.txt";
64     int count_to_position=0;
65     Input = fopen("in.txt", "rt");
66     Output = fopen(TemporaryFileName, "wt");
67
68     if(Input == NULL)
69     {
70         printf("Empty or missing file!");
71         exit(1);
72     }

```

Figure 6: replace specific instance

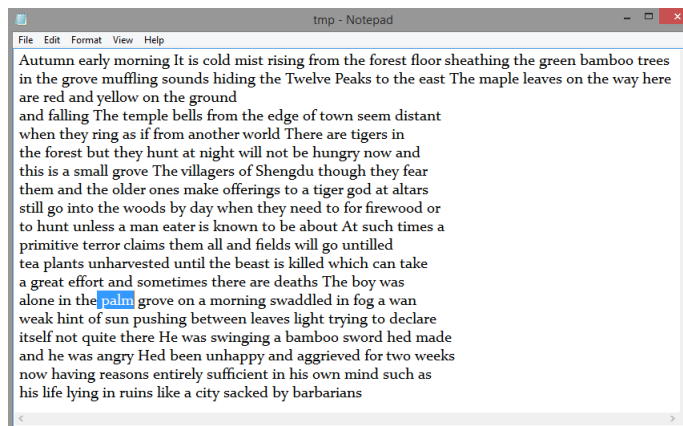


Figure 7: replace specific instance



```

"C:\Users\User\Desktop\PROJECT\sorted list.exe"
Autumn -> 1 occurrence
early -> 1 occurrence
it -> 1 occurrence
cold -> 1 occurrence
mist -> 1 occurrence
rising -> 1 occurrence
floor -> 1 occurrence
sheathing -> 1 occurrence
green -> 1 occurrence
trees -> 1 occurrence
muffling -> 1 occurrence
sounds -> 1 occurrence
hiding -> 1 occurrence
twelve -> 1 occurrence
tears -> 1 occurrence
east -> 1 occurrence
maple -> 1 occurrence
way -> 1 occurrence
here -> 1 occurrence
red -> 1 occurrence
yellow -> 1 occurrence
ground -> 1 occurrence
falling -> 1 occurrence
temple -> 1 occurrence

int sorted_list()
{
    struct word_occurrence occ[1000];
    int n,i=0,j; /* n = number of words*/
    int aux; // a place-holder for swapping the value of two word occurrences
    char auxword[50];
    char word[20]; // a string which helps us swap two words
    char s[2000];
    FILE *Input;
    Input = fopen("in.txt", "rt");
    if(Input == NULL)
    {
        printf("Missing or empty file!");
        exit(0);
    }
}

```

Figure 8: sorted list

```

//REMOVE VOWELS AND LEAVE ONLY CONSONANTS
#include <stdio.h>
#include <string.h>
FILE *f;
int check_consonants(char c)
{
    switch(c) {
        case 'b':
        case 'c':
        case 'd':
        case 'f':
        case 'g':
    }
}

First line after deleting consonants: Au ea oi l i o i i i o e oo eai o ee
o ee

Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.

return 0;
fclose(f);
getch();
}

```

Figure 9: delete consonants