”TEXT EDITOR LIBRARY”

Jianu Laura

Anul I

Calculatoare română

Grupa 10104 A

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**(**stricmp**$(s, word) == 0)then$
6. **return** 1
7. **return** 0
8. Close file "in.txt"

OCCURRENCE(s, word)

1.$counter \leftarrow 0$
2. Open file "in.txt"
3. **if** f == NULL then
4. ▷ "Missing or empty file!"
5. **while**(reading s until EOF)
6. **if**(**stricmp**$(s, word) == 0)then$
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

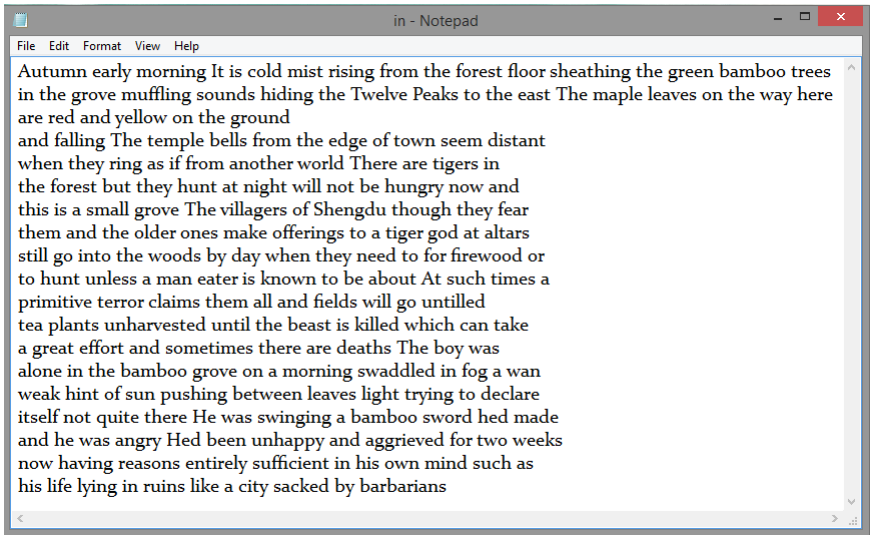

Figure 1: Input

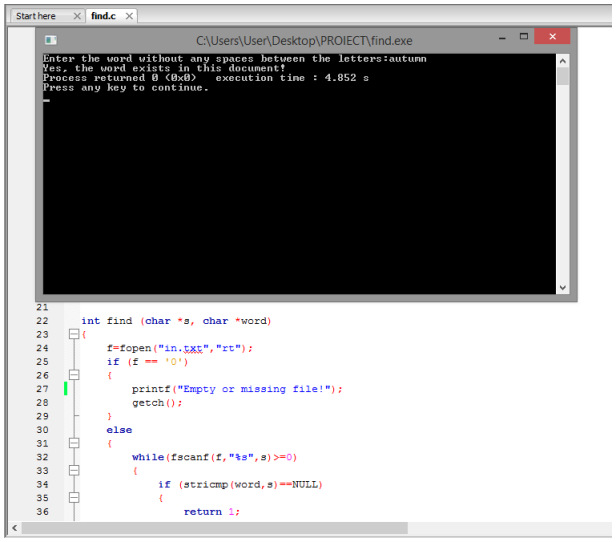

Figure 2: find

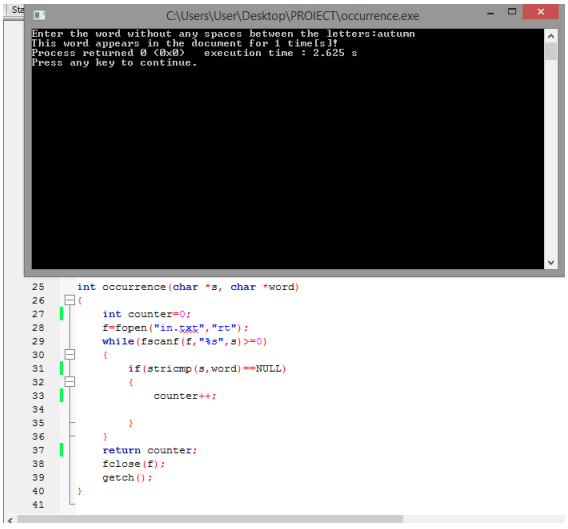

Figure 3: occurrence

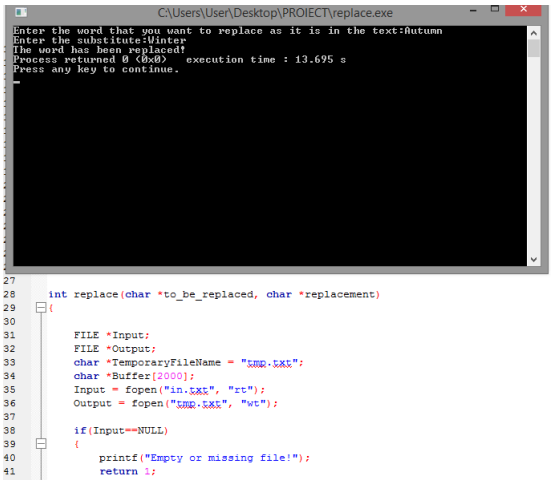

Figure 4: replace

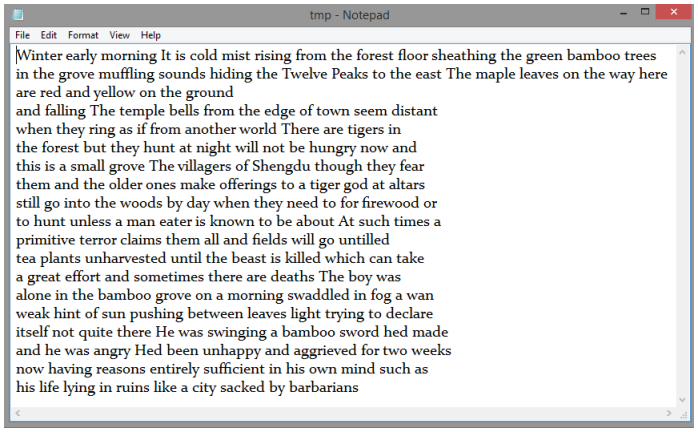

Figure 5: replace

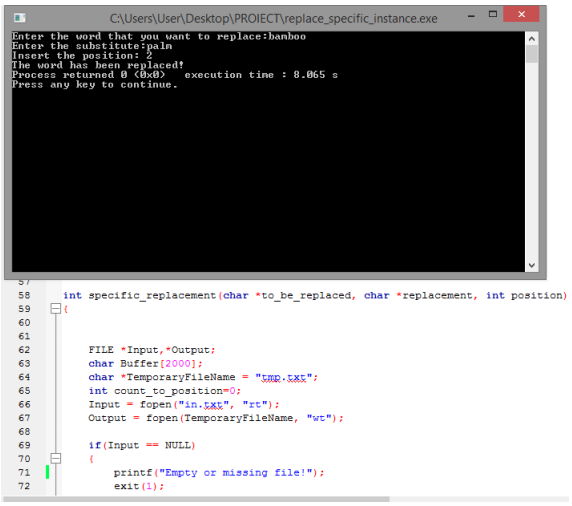

Figure 6: replace specific instance

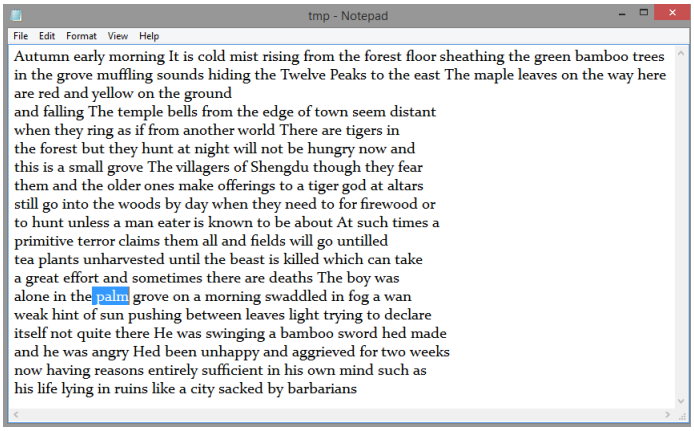

Figure 7: replace specific instance

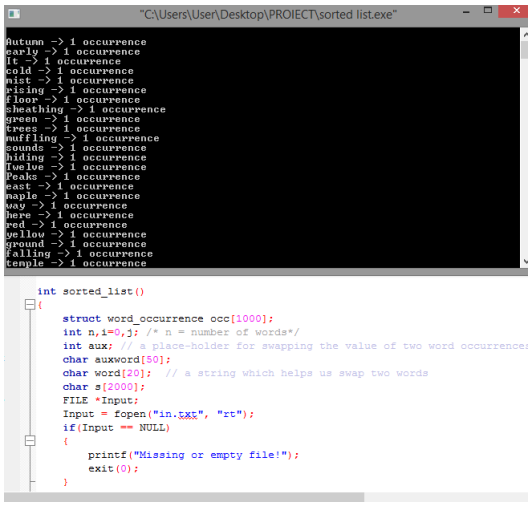

Figure 8: sorted list

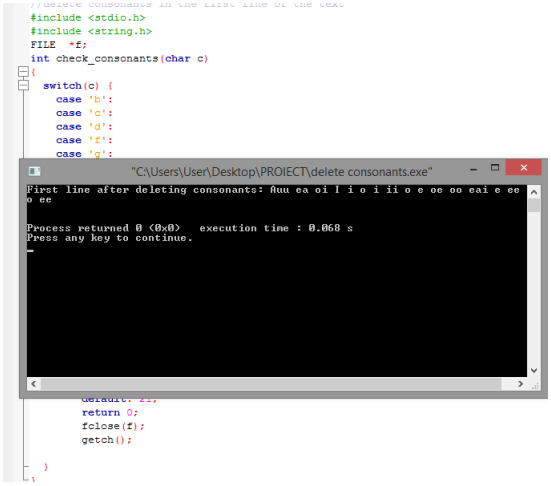

Figure 9: delete consonants