PROGRAMLAMA DİLLERİ
3. ÖDEV

Öğrenci Adı: Osman Araz

Öğrenci Numarası: 16011020

Öğretmen: M. Amaç Güvensan

Ödev Teslim Tarihi: 10.12.2017

Bilgisayar Bilgileri: 4 çekirdekli 64 bit işlemci, 8 GB RAM

Program Derleyicisi: GCC 7.2.0

Geliştirme Ortamı: Manjaro KDE Edition 17.0.6

Kelime Düzeltici

Bir dosyadan okunan kelimeleri başka bir dosyaya, eğer kelime doğru yazılmışsa direkt olarak; değilse alternatiflerini yazdıran program.

Çözüm

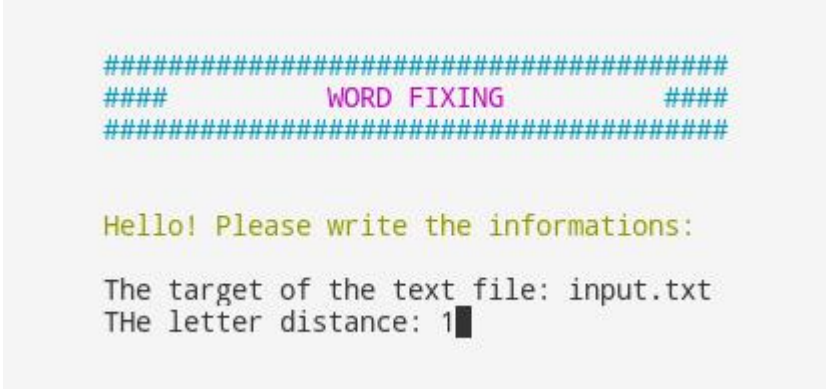
Programın başında Türkçedeki 28 adet harfi (“ğ” harfi hariç) simgeleyen 28 adet string dizisini barındıran bir çok boyutlu dizi oluşturulur. Bu diziye 28 adet kelime dosyasından tüm kelimeler ilk harflerine göre eklenir. Daha sonra kullanıcıdan alınan girdi dosyasındaki kelimeler, istenen hasassiyete göre kelime dizisinde aranır. Eğer aranan kelime bulunursa dosyaya direkt olarak yazdırılır. Eğer bulunamazsa aranan kelimeye benzer kelimeler, benzerliklerine göre sıralı şekilde dosyaya yazdırılır.

Ekran Görüntüleri:

Girdi dosyasının içeriği:


```
zeyzin
ada
aca
zil
nakarat
aile
vagon
maden filizi
makeni
sağlık
canacar
kahte
çaçaşır
şırret
çapel
yayuk
katap
muzaffar
```

Girdi dosyasının konumu ve istenen kelime hassasiyeti kullanıcıdan alınır:



İstenen hassasiyete göre çıktı dosyası oluşturulur:

```
########################################
####          WORD FIXING           ####
########################################


Search completed!
Please check the output file: "output.txt".
```

Çıktı dosyası:


```
- zeytin
+ ada
- uca aza aya ata asa ara ana ama ama ala ala aka aha ağa ada acı aba
+ zil
+ nakarat
+ aile
+ vagon
+ maden filizi
- madeni
+ sağlık
- canavar
- sahte kahve Kahta kahpe
- çamaşır
- şirret
- papel çepel çapul
- yayık yatuk yamuk
- kitap katip katar kasap
- muzaffer
```

Kelime hassasiyeti arttırılır:

```
########################################
####          WORD FIXING           ####
########################################


Hello! Please write the informations:

The target of the text file: input.txt
THe letter distance: 2
```

```
########################################
####          WORD FIXING           ####
########################################


Search completed!
Please check the output file: "output.txt".
```

Çıktı dosyası:


```
- zeytin zerrin zeplin zengin neyzen benzin
+ ada
- uca aza aya ata asa ara ana ama ama ala ala aka aha ağa ada acı aba Ula şua Şia öcü oya ova ora ona oma
oha oda oba ita İsa ima ila ika ifa ıra eza ela eda ece dua boa azı ayn ayı aut ati aşk aşı ast ask asi
arz art arş arp ark Ari arı ant ani anı alt alp alo ali alg akü aks akı ait ahu Ahi ahi ağı agu aft afi
adi açı abu
+ zil
+ nakarat
+ aile
+ vagon
+ maden filizi
- madeni medeni maşeri Maruni manevi makine
+ sağlık
- canavar car car
- sahte kahve Kahta kahpe tahta şahne sahre sahne rahne rahle mahfe külte köhne köfte korte kerte kehle
kayme katre kasti karye karne kanto kaime kaide kahya kahir kahin kahil kahır kaffe Bahçe bahçe
- çamaşır yaraşır
- şirret şöhret şirket şerbet
- papel çepel çapul panel nipel lamel Hamel halel gazel çökel çipil çepez çeper çekel çatal çaput çapma
çaplı çapla çapar çapak çakıl çakal
- yayık yatuk yamuk yumuk yoluk yazık yayma yaylı yayla yayış yayın yayım yaygı yayan yavuz yatık yatak
yasak yarık yarak yapık yapak yanık yanak yamak yalak Yakut yakut yahut tavuk şayak suyuk pamuk natuk
maşuk layık kayık kayak kavuk kabuk hoyuk gayur dayak çabuk ayyuk
- kitap katip katar kasap yatay yatak vatan Tatar tatar şataf şarap şahap ratıp patak matah lakap kutup
kutan kotan kıtal ketal Keşap kebap kazaz Kazan kazan Kazak kazak kayıp kayar kayan kayak kayaç kavat
kavas kaval Kavak kavak kavaf katre katot katma katlı katkı katil kat'i katır katım katık kaşar kaşan
Karay karar kapan kapak kanat kanal Kaman kalıp kalay kalas Kalan kalan kalak kakao kakaç kağan kadar
kaçar kaçak kaban kabak hitap hatip Hatay harap gazap çatal Çatak çatak çalap bitap batar batak
- muzaffer muvaffak
```

C Kodu:


```c
/*
    ATTENTION: This code designed for Linux, may not works correctly in Windows.
*/

/*
    ATTENTION: The input file and the speller files must encoded as UTF-8.
*/

/*
    Programming Languages - Autumn 2017 - Assignment 3

    A program that fixes the wrong words within the given file.

    @author
    Name: Osman Araz
    Student NO: 16011020
    Date: 10.12.2017
    E-Mail: arazosman@outlook.com
    Compiler Used: GCC 7.2.0
    Computer Hardware: 64 bit Quad Core CPU, 8 GB RAM
    IDE: None (Visual Studio Code used as a text editor.)
    Operating System: Manjaro KDE Edition 17.0.6
*/

#include <stdio.h>
#include <stdlib.h>   // used for malloc(), realloc() and free() functions
#include <stdbool.h>  // used for true and false boolean values
#include <limits.h>   // used for INT_MAX value
#include <locale.h>   // used for setlocale() function
#include <wchar.h>    // used for working correctly on special Turkish characters
#include <wctype.h>   // used for towupper() and towlower() functions

// ANSI color defining
#define COLOR_RED     "\x1b[31m"
#define COLOR_GREEN   "\x1b[32m"
#define COLOR_YELLOW  "\x1b[33m"
#define COLOR_BLUE    "\x1b[34m"
#define COLOR_MAGENTA "\x1b[35m"
#define COLOR_CYAN    "\x1b[36m"
#define COLOR_RESET   "\x1b[0m"

/*
    ABOUT WIDE CHARS:

    Wide chars used on whole the program. They are the only way I found to work correctly
    on non-ASCII Turkish characters in Linux. Some of the syntaxes, functions and their
    explanations about the wide chars:

    wchar_t : represents a wide char variable
    wchar_t * : represents an array of wide chars (string of wide chars)
    wcslen() : a function to compute the length of a string of wide chars
    wcscpy() : a function to assign a string of wide chars to another
    wcstombs() : a function to convert a wide char string to a normal char string
    towlower() : a function to make lowercase a wide char
    towupper() : a function to make uppercase a wide char
    fgetws() : a function to reading wide chars from a file
    fputws() : a function to writing wide chars to a file

    NOTE: The initializing of a wide char/string of wide chars must prefixed by
    letter of "L". Examples:

    wchar_t ch = L'ç';
    wchar_t *str = L"abcçdef";
*/

wchar_t ***words; // the 2-D string array of spelling files
const wchar_t *trLetters = L"ABCÇDEFGHIİJKLMNOÖPRSŞTUÜVYZ"; // Turkish characters (except "Ğ")
const int trSize = 28; // the number of Turkish characters (except "Ğ")

/*
    Main function: It gets the source file location and the number of letter distance
```

```
72      from the user.
73  */
74  void printBanner();
75  void crashFile();
76  void crashMemory();
77  int main()
78  {
79      void getWords();
80      void searchWord(wchar_t *, int, FILE *);
81
82      setlocale(LC_ALL, ""); // used for display correctly the special Turskish characters
83      printBanner(); // printing the banner of the program
84      char fileName[50];
85      int i, j, dist;
86
87      getWords(); // saving the words to the array from the spelling files
88
89      // getting informations from the user:
90      printf(COLOR_YELLOW "\tHello! Please write the informations: \n\n" COLOR_RESET);
91      printf("\tThe target of the text file: ");
92      scanf("%s", fileName);
93      printf("\tTHe letter distance: ");
94      scanf("%d", &dist);
95
96      // opening input and output files
97      FILE *fi, *fo;
98      fi = fopen(fileName, "r");
99      fo = fopen("output.txt", "w");
100
101     if (!fi || !fo) // checking crashes
102         crashFile();
103
104     wchar_t word[100];
105
106     while (fgetws(word, 100, fi)) // reading words from the input file
107         searchWord(word, dist, fo); // searching the word on the array of the saved words
108
109     printBanner();
110     printf(COLOR_YELLOW "\tSearch completed!\n\tPlease check the output file: \"output.txt\".\n\n"
        COLOR_RESET);
111
112     // deallocation the array of the saved words
113     for (i = 0; i < trSize; i++)
114         free(words[i]);
115
116     free(words);
117
118     // closing the input and output files
119     fclose(fi);
120     fclose(fo);
121
122     return 0;
123 }
124
125 /*
126     Function which saves the words from the spelling files to the array.
127 */
128 char *getFileTarget(wchar_t);
129 void getWords()
130 {
131     int i, p;
132     words = (wchar_t ***)malloc(trSize*sizeof(wchar_t **));
133
134     if (!words) // checking crashes
135         crashMemory();
136
137     wchar_t tmp[100];
138
139     for (i = 0; i < trSize; i++) // for all 28 letters
140     {
141         int j = 0, k = 1000;
```

```
142         words[i] = (wchar_t **)malloc(k*sizeof(wchar_t *));
143 
144         if (!words[i]) // checking crashes
145             crashMemory();
146 
147         // getting the file location according to the letter and then opening the file
148         char *fileTarget = getFileTarget(trLetters[i]);
149         FILE *file = fopen(fileTarget, "r");
150 
151         if (!file) // checking crashes
152             crashFile();
153 
154         for (p = 0; p < k; p++)
155         {
156             // memory allocation for strings
157             words[i][p] = (wchar_t *)malloc(30*sizeof(wchar_t));
158 
159             if (!words[i][p]) // checking crashes
160                 crashMemory();
161         }
162 
163         while (!feof(file))
164         {
165             while (j < k && !feof(file))
166             {
167                 fgetws(tmp, 100, file); // getting words from the file
168                 wcscpy(words[i][j], tmp); // copying words to the array
169                 j++;
170             }
171 
172             if (!feof(file))
173             {
174                 k *= 2; // if any words remained, then the size of the array extends
175                 words[i] = realloc(words[i], k*sizeof(wchar_t *));
176 
177                 if (!words[i]) // checking crashes
178                     crashMemory();
179 
180                 for (p = k/2; p < k; p++)
181                 {
182                     // memory allocation for strings (for extended parts)
183                     words[i][p] = (wchar_t *)malloc(30*sizeof(wchar_t));
184 
185                     if (!words[i][p]) // checking crashes
186                         crashMemory();
187                 }
188             }
189         }
190 
191         fclose(file); // closing the file
192     }
193 }
194 
195 /*
196     Function which searches the word on the array of the saved words.
197     @param word: the word which will be searched
198     @param dist: the letter distance
199     @fo: the output file
200 */
201 int compareStrings(wchar_t *, wchar_t *, int);
202 void addSimilarWord(wchar_t **, wchar_t *, wchar_t *, int);
203 void searchWord(wchar_t *word, int dist, FILE *fo)
204 {
205     // count: represents the number of similar words
206     // capacity: represents the capacity of similar words
207     // isFound: represents does the word found on the speller files or not
208     int i, j, p, count = 0, capacity = 100;
209     bool isFound = false;
210 
211     // array of the similar words
212     wchar_t **similarWords = (wchar_t **)malloc(capacity*sizeof(wchar_t *));
```

```
213
214     if (!similarWords) // checking crashes
215         crashMemory();
216
217     for (i = 0; i < capacity; i++)
218     {
219         // memory allocation for similar strings
220         similarWords[i] = (wchar_t *)malloc(30*sizeof(wchar_t));
221
222         if (!similarWords[i]) // checking crashes
223             crashMemory();
224     }
225
226     // the word will be searched on all the 28 speller files until it found
227     for (i = 0; i < trSize && !isFound; i++)
228     {
229         j = 0;
230
231         // a while loop to search the word on the specific speller file
232         // if there is a string, wcslen() will be different than 0
233         while (!isFound && wcslen(words[i][j]))
234         {
235             // diff: represents the letter distance between the word and the saved words
236             int diff = compareStrings(word, words[i][j], dist);
237
238             if (diff == 0) // if letter distance is 0, then the word is found
239             {
240                 // writing the word on the output file
241                 fputws(L"+ ", fo);
242                 fputws(word, fo);
243                 isFound = true;
244             }
245             // if letter distance is greater than wanted letter distance, then diff will be -1
246             else if (diff != -1)
247             {
248                 if (count >= capacity)
249                 {
250                     // if count is come to the capacity, then capacity will be increase
251                     capacity *= 2;
252                     similarWords = realloc(similarWords, capacity*sizeof(wchar_t *));
253
254                     if (!similarWords) // checking crashes
255                         crashMemory();
256
257                     for (p = capacity/2; p < capacity; p++)
258                     {
259                         // memory allocation for similar strings (for extended parts)
260                         similarWords[p] = (wchar_t *)malloc(30*sizeof(wchar_t));
261
262                         if (!similarWords[p]) // checking crashes
263                             crashMemory();
264                     }
265                 }
266
267                 // the found similar word will be added on the array of the similar words
268                 addSimilarWord(similarWords, word, words[i][j], count);
269                 count++;
270             }
271
272             j++;
273         }
274     }
275
276     // if the word doesn't found on the saved words, then the similar words
277     // will be written on the output file
278     if (!isFound)
279     {
280         fputws(L"- ", fo);
281
282         for (i = 0; i < count; i++)
283         {
```

```
284             fputws(similarWords[i], fo);
285             fputws(L" ", fo);
286         }
287     }
288 
289     fputws(L"\n", fo);
290 
291     // deallocation the array of the similar words
292     for (i = 0; i < capacity; i++)
293         free(similarWords[i]);
294 
295     free(similarWords);
296 }
297 
298 /*
299     Function which adds the similar words to the array of the similar words
300     according to their letter distances between the searched word.
301     @param **similarWords: the array of the similar words
302     @param *word: the word which searched on the array of the saved words
303     @param *fileWord: the word which will be added to the array of the similar words
304     @param count: the count of the array of the similar words
305 */
306 void addSimilarWord(wchar_t **similarWords, wchar_t *word, wchar_t *fileWord, int count)
307 {
308     int i, j = 0;
309 
310     // the similar word will be insert to the array of the similar words
311     // according to its letter distance between the searched word
312     while (j < count && compareStrings(fileWord, word, INT_MAX) > compareStrings(similarWords[j],
    word, INT_MAX))
313         j++;
314 
315     for (i = count; i > j; i--)
316         wcscpy(similarWords[i], similarWords[i-1]); // similar words are shifting to the right
317 
318     wcscpy(similarWords[j], fileWord);
319 }
320 
321 /*
322     Function which compares the letter distances between two strings.
323     @param *s1: a string
324     @param *s2: another string
325     @param dist: the maximum letter distance which permitted
326     @return diff: the letter distance between given strings, if two strings have different lentghs
327     or letter distance is bigger than the maximum letter distance, then it returns -1
328 */
329 int compareStrings(wchar_t *s1, wchar_t *s2, int dist)
330 {
331     // ignoring the new line characters on the end of the strings
332     if (s1[wcslen(s1)-1] == '\n' || s1[wcslen(s1)-1] == ' ')
333         s1[wcslen(s1)-1] = '\0';
334 
335     if (s2[wcslen(s2)-1] == '\n' || s2[wcslen(s2)-1] == ' ')
336         s2[wcslen(s2)-1] = '\0';
337 
338     // computing the length of the strings
339     int s1_length = wcslen(s1), s2_length = wcslen(s2);
340 
341     // if the length of the strings are unequal, then it returns -1
342     if (s1_length != s2_length)
343         return -1;
344 
345     int diff = 0, j = 0;
346 
347     // computing the letter distance
348     while (j < s1_length && diff <= dist)
349     {
350         if (towlower(s1[j]) != towlower(s2[j]))
351             diff++;
352 
353         j++;
```

```
354     }
355
356     // if letter distance is excesses the maximum letter distance, then it returns -1
357     if (diff > dist)
358         return -1;
359
360     return diff;
361 }
362
363 /*
364     Function which generates the file target.
365     @param ch: the fifth char of the target of "imla/?.txt"
366     @return fileTarget: the generated file target
367 */
368 char *getFileTarget(wchar_t ch)
369 {
370     char *fileTarget = (char *)malloc(20*sizeof(char));
371     wchar_t targetWchar[] = L"imla/?.TXT";
372     targetWchar[5] = towupper(ch);
373     wcstombs(fileTarget, targetWchar, 20); // wchar_t * > char *
374
375     return fileTarget;
376 }
377
378 /*
379     Function which displays the error message about the file opening.
380 */
381 void crashFile()
382 {
383     printBanner();
384     printf(COLOR_RED "\tFile could not be opened.\n\n" COLOR_RESET);
385     exit(1);
386 }
387
388 /*
389     Function which displays the error message about the memory allocation.
390 */
391 void crashMemory()
392 {
393     printBanner();
394     printf(COLOR_RED "\tNot enough space.\n\n" COLOR_RESET);
395     exit(1);
396 }
397
398 /*
399     Function which prints the banner of the program.
400 */
401 void printBanner()
402 {
403     system("clear"); // cleaning the screen
404     printf(COLOR_CYAN "\n\t########################################\n");
405     printf("\t####" COLOR_MAGENTA "          WORD FIXING          " COLOR_CYAN "####\n");
406     printf("\t########################################\n\n\n" COLOR_RESET);
407 }
```