19.12.2020

YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM3021 ALGORİTMA ANALİZİ 3.ÖDEV-2.SORU RAPORU

Sorgulanan Bir Cümlede Yanlış Yazılmış Kelimelerin Yerine Doğru Kelimeler Öneren Sistem Tasarımı

AHMET SAİD SAĞLAM

17011501

KONU

Yanlış Yazılan Kelimeler Yerine Doğru Kelimeler Öneren Sistem Tasarımı

Bu ödevde, sözlük olarak verilen dokümandaki kelimelerin hashing yöntemi ile bir hash tablosuna yerleştirilmesi ve bir cümle sorgulandığında, cümledeki kelimeler yine hashing yöntemi ile tabloda aranarak içinde yanlış yazılan kelimeler yerine tablodan kelime önerisi yapılması istenmiştir.

ÇÖZÜM

Kütüphane Eklenmesi ve Makro Atanması

```c
/*
@file
BLM3021 2020-2021 GUZ ODEV-3 / Soru-2
Bu programda sorgulanan bir cümlede, yanlış yazılmış kelimelerin yerine doğru kelimeler öneren bir sistem tasarımı gerçeklenmiştir

@author
İsim: Ahmet Said SAĞLAM
Öğrenci No: 17011501
Tarih: 19.12.2020
E-Mail: l1117501@std.yildiz.edu.tr
Compiler: TDM-GCC 4.9.2 64 bit-Release
IDE: DEV-C++ (version 5.11)
İşletim Sistemi: Windows 10 Pro 64 bit
*/

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

#define SENTENCE_SIZE 200   //kullanıcıdan alınacak olan maksimum cümle uzunlugu
#define BUFFER_SIZE 10000 //txt dosyadan alınan satırın saklanacağı bufferin boyutu
#define TEXT_NAME_SIZE 20 // max file ismi uzunlugu
#define AYRAC " "    //kelimelerin ayrilacagi delim ifadesi
#define HORNER_NUMBER 7 //horner numarasi hesaplanirken kullanilcak olan asal sayi
#define WORD_SIZE 20    //alinan bir kelimenin maksimum uzunlugu
#define MM 996 //double hashing hesabinda, iki numarali keyi hesaplarken kullanilacak olan sayi
#define M 997 //hash tablosunun boyutu
#define SOZLUK_TXT "sozluk.txt" //sozlugu tutan hash tablosunun kaydedildigi dosyanin ismi
#define HATALI_TXT "hatalikelime.txt"   //hatali kelime sozlugunu tutan hash tablosunun kaydedildigi dosyanin ismi
#define K 2 //Maksimum olabilecek Edit Distance
```

Kodun ilk kısmında gerekli olabilecek kütüphaneler eklenmiş, kodda kullanılmak üzere gerekli makrolar atanmış ve programın geliştiricisi ve geliştirilip çalıştırıldığı ortam hakkında bilgiler verilmiştir.

Yapı Tanımları ve clear_input_buffer Fonksiyonu

```c
//sozlukte tutulacak kelimeler icin structure
typedef struct node {
    int count;  //goz bos ise 0, dolu ise 1 degerini alir
    float loadFactor;   //tablonun doluluk orani
    char word[WORD_SIZE];   //tutulan kelime
} node;

//hatali kelime sozlugunde tutulacak olan elemanlar icin structure
typedef struct node_2 {
    int count;  //goz bos ise 0, dolu ise 1 degerini alir
    float loadFactor;   //tablonun doluluk orani
    char word[WORD_SIZE];   //tutulan hatalı kelime
    char onerilen_kelime[WORD_SIZE]; //önerilen kelime
} node_2;

//gets() fonksiyonu kullanılmadan önce eger scanf kullanıldıysa gets() duzgun calismayacagi icin input buffer'i temizlemmek icin gerekli fonksiyon
int clear_input_buffer(void) {
    int ch;
    while (((ch = getchar()) != EOF) && (ch != '\n'));
    return ch;
}
```

struct node yapısı sözlükteki orijinal kelimelerin okunduğu hash tablosunun her bir gözünde bulunan yapıdır. Yapıdaki count değeri gözün dolu olup olmadığını, loadFactor değeri hash tablosunun doluluk oranını, word dizisi ise okunan kelimeyi tutar.

struct node_2 yapısı sözlükte bulunamayan hatalı kelimeleri tutan hatalı kelime sözlüğünün hash tablosunun her bir gözünde bulunan yapıdır. Yapıdaki count değeri gözün dolu olup olmadığını, loadFactor değeri hash tablosunun doluluk oranını, word dizisi okunan hatalı kelimeyi, onerilen_kelime dizisi ise bu hatalı kelime yerine daha önceden önerilip kullanıcının kabul ettiği geçerli kelimeyi tutar.

clear_input_buffer fonksiyonu, scanf() fonksiyonundan sonra gets() fonksiyonu kullanılacağından input buffer'ı temizlemek için kullanılır. Aksi taktirde gets() fonksiyonu input buffer'da scanf() fonksiyonundan kalan ENTER girişini ilk olarak okuyacak ve istenen sonucu vermeyecektir.

getHorner Fonksiyonu

```c
//icine verilen kelimenin horner sayisini donduren fonksiyon
long long getHorner(char *word) {
    long long key = 0L; //kelimenin sayisal key karşılığını tutacak olan degisken
    int i; //dongu degiskeni
    //kelimenin horner sayisi hesaplanir
    for(i = 0; i < strlen(word); i++) {
        key = (long long) (HORNER_NUMBER * key + (word[i] - 'a' + 1));
    }
    return (long long) key;

}
```

getHorner fonksiyonu içerisine bir kelime alır ve Horner's Method'a göre bu kelimenin sayısal karşılığını üretip dışarıya döndürür.

min, min_3 ve diff Fonksiyonları

```
69  //icine verilen sayilardan kucuk olanı donduren fonksiyon
70  int min(int x, int y) {
71      if(x < y) {
72          return x;
73      }
74      return y;
75  }
76
77  //icine aldıgı 3 sayıdan en kucugunu donduren fonksiyon
78  int min_3(int x, int y, int z) {
79      return min(min(x,y), z);
80  }
81
82  //icine aldıgı iki karakter farklıysa 1, aynıysa 0 donduren fonksiyon
83  int diff(char x, char y) {
84      if(x == y) {
85          return 0;
86      }
87      return 1;
88  }
```

min fonksiyonu içerisine aldığı iki sayıdan küçük olanı, min_3 fonksiyonu ise içerisine aldığı 3 sayıdan en küçük olanı döndürür. diff fonksiyonu içerisine aldığı iki karakter aynı ise 0, farklı ise 1 döndürür.

calculateDistance Fonksiyonu

```
90  //icerisine aldıgı iki kelimenin L.Edit Distance'nı hesaplayan fonksiyon
91  int calculateDistance(char *word_1, char *word_2, int w1_length, int w2_length) {
92      //Edit Distance Kesin Olarak 2'den büyükse(kelimelerin uzunlukları farkı 2'den büyükse kesin olarak 2'den büyüktür) hesaplama yapmadan fonksiyondan çıkılır.
93      if(w1_length - w2_length > K || w2_length - w1_length > K) {
94          return 99;  //distance olarak 99 dondurulur. Anlamsız ve kullanılmayacak bir degerdir
95      }
96      int ED[w1_length + 1][w2_length + 1]; //Edit Distance hesabında kullanılan matris olusturulur
97      int i, j; //dongu degiskenleri
98      int control = 0;    //ED kesin olarak 2'den buyuk mu diye kontrolun yapılacagı degisken
99
```

calculateDistance fonksiyonu içerisine aldığı iki kelimenin Edit Distance'ını dinamik hesaplama yöntemi ile hesaplar. Fonksiyon içerisine iki kelime ile birlikte bu kelimelerin uzunluklarını alır. İlk adımda kelimelerin uzunlukları farkı hesaplanır ve bu fark Edit Distance için belirlenen eşik değerinden fazlaysa, Edit Distance da minimum kelimelerin uzunluk farklarına eşit olacağından fonksiyondan çıkılır.

Eğer ki uzunluk farkları eşik değerinin altındaysa, dinamik hesaplamada kullanılacak matris ile beraber döngü değişkenleri ve control değişkeni tanımlanır. control değişkeni hesaplama sırasında matrisin ilgili satırındaki eşik değerinden fazla olan göz sayısını tutar. Bu sayı matrisin sütun sayısına eşit olduğu anda hesaplamalar durdurulur. Bunun nedeni matrisin bir alt satırında, üst satırdakinden daha küçük değerler tutulamayacak olması ve Edit Distance'a ancak son satırın son sütun değeri ile ulaşılabilmesidir. Böylece, Edit Distance'ın, eşik değerini geçeceği kesinleşmiş olur.

```
    i = 0;  //i ilklendirilir
    //i satır sayısı kadar donsun ve control sutun sayisindan kucuk ise donsun
    while(i <= w1_length && control <= w2_length) {
        control = 0;    //her satir icin control 0'lanır
        //satırda sutun sayısı kadar dön matrisin gözlerine eris
        for(j = 0; j <= w2_length; j++) {
            //matrisin 0. satırını ilklendir
            if(i == 0) {
                ED[i][j] = j;
            }
            //matrisin 0. sutunu ilklendirilir
            else if(j == 0) {
                ED[i][j] = i;
            }
            //kelimelerin ilgili harfinde insert-delete-replace işlemlerinden en az mesafe hangisi ise o seçilir ve matris doldurulur.
            else {
                ED[i][j] = min_3((ED[i-1][j] + 1), (ED[i][j-1] + 1), (ED[i-1][j-1] + diff(word_1[i-1],word_2[j-1])));
            }
            //matrisin ilgili satirindaki ilgili gözün değeri 2'den büyükse control degiskenini 1 arttır
            if(ED[i][j] > K) {
                control++;
            }
            //printf("kontrol - control : %d------satir : %d -- sutun : %d\n",control,i,j);
        }
        i++;    //yeni satıra gecmek icin i'yi guncelle
    }
```

Hesaplamalar bir while döngüsü yardımıyla satırlar gezilerek ve aynı anda control değişkeni kontrol edilerek ve bir for döngüsü yardımıyla sütunlar gezilerek yapılır. For döngüsünün içinde matrisin ilk satır ve sütun değerlerinin ilklendirmelerinin yanı sıra matrisin içeriği de doldurulmaya başlanır. Kelimelerin ilgili harflerinde, insert, delete ve replace işlemlerinden hangisi daha az distance' a sebep olacaksa seçilir ve matris böylece doldurulmuş olur.

```
    //while dongusunden matrisin sonuna gelindigi icin cıkıldıysa, Edit Distance degeri disari dondurulur
    if(control <= w2_length) {
        return ED[w1_length][w2_length];    //Edit Distance dışarı dondurulur
    }
    //while dongusunden, control sutun sayısına esit oldugu icin cikildiysa, matrisin o satırındaki tum degerler 2'den buyuk demektir
    //bir alt satıra inecek olan tum degerler 2'den buyuk olacagı icin Edit Distance'ın 2'den buyuk olacağı kesinlesmistir.
    else {
        return 99; //distance olarak 99 dondurulur. Anlamsız ve kullanılmayacak bir degerdir
    }
}
```

Fonksiyonun son kısmında while döngüsünün çıkış nedeni kontrol edilir. Eğer matrisin sonuna gelindiği için çıkıldıysa Edit Distance değeri dışarı döndürülür. Diğer durumda ise Edit Distance değeri olamayacak bir değer dışarı döndürülür ve fonksiyon işlemini tamamlar.

totalDistances Fonksiyonu

```
//içerisine aldığı kelime ile sözlükteki kelimelerin mesafelerini hesaplar ve dizide kaydeder. Diziyi dışarı döndürür.
int *totalDistances(char *word, struct node *sozluk) {
    int *distances = (int*) malloc(M * sizeof(int));    //sözlükteki kelimelerin, fonksiyonun içine aldığı kelimeye göre ED'sini tutan dizi
    int i;  //dongu degiskeni
    for (i = 0; i < M; i++) {
        //hash tablosunda(sozlukte) ilgili göz boş ise
        if(sozluk[i].count == 0) {
            distances[i] = 99;  //distance olarak 99 atanır. Anlamsız bir değerdir.
        }
        else {
            distances[i] = calculateDistance(word, sozluk[i].word, strlen(word), strlen(sozluk[i].word)); //ilgili ED hesaplanır
        }
    }
    return distances;//dizi dışarı döndürülür
}
```

totalDistances fonksiyonu içerisine aldığı kelimenin sözlükteki diğer tüm kelimelere göre Edit Distance'ını hesaplar ve bir dizide tutar. Dizinin indisleri kelimelerin sözlükteki adreslerini temsil ederken o indisteki değer ise ED değerini temsil eder. Sözlüğün ilgili adresinde kelime yoksa Edit Distance değeri olarak Edit Distance değeri olamayacak bir değer diziye atanır. Son adımda ise dizi dışarı döndürülür ve işlemler tamamlanır.

levenshtein Fonksiyonu

```c
//cumleyi kullanıcıdan alarak gerektiginde oneri yapan ve cumlenin son halini ekrana yazdıran fonksiyon
void levenshtein(struct node *sozluk, struct node_2 *hatali_sozluk, int *hatali_count) {
    int count = 0;  //cumlede kac adet kelime var tutan degisken
    int oneri_count = 0;    //onerilebilecek kelime sayısı
    char *sentence = (char*) calloc(SENTENCE_SIZE, sizeof(char));   //cumle için yer açılır
    char **words = (char**) calloc(1, sizeof(char*));   //cumleden parcalanan kelimeleri tutmak için matris
    words[count] = (char*) calloc(WORD_SIZE, sizeof(char)); //matrisin her satırında kelime tutmak için yer açılır
    char **oneriler = (char**) calloc(1, sizeof(char*)); //ED'si 1 veya 2 olan onerilecek kelimeleri tutmak için matris
    oneriler[oneri_count] = (char*) calloc(WORD_SIZE, sizeof(char));    //matrisin her satırında kelime tutmak için yer açılır
    char *kullanici_onerisi = (char*) calloc(WORD_SIZE, sizeof(char));  //kullanıcı önerilen kelimeler arasından seçim yapar. Bu seçimi tutan dizi için yer açılır
    int cont = 1; //yeni cumle girisini kontrol eden degisken
    int isExist; //kelime sozluk için tanımlanan hash table'da var mi yok mu donus degerini tutan degisken(kelime tabloda mevcutsa 1, degilse 0)
    int isExist_2; //kelime hatali kelime sozlgu için tanımlanan hash table'da var mi yok mu donus degerini tutan degisken(kelime tabloda mevcutsa adresi, yoksa 1000)
    int j,i;    //dongu degiskenleri
    int *distances; //Edit distance'ları tutan dizi için tanımlama
    char *token;    //strtok ile alınan kelime
```

levenshtein fonksiyonu içerisine sözlük ve hatalı kelime sözlüğü hash tablolarıyla beraber hatalı kelime sözlüğü tablosunun eleman sayısını alır. Fonksiyon içerisinde ilk olarak gerekli matrisler, diziler ve değişkenler tanımlanır. Her bir değişkenin ne işe yaradığı yorum satırlarıyla açıklanmıştır.

```c
    //kullanıcı istedikçe dönen while
    while(cont) {
        count = 0;  //her yeni cümlede yeni aramalar icin count değerleri 0'lanır
        oneri_count = 0;    //her yeni cümlede yeni aramalar icin count değerleri 0'lanır
        printf("Cumleyi giriniz...\n");
        gets(sentence); //cümle kullanıcıdan alınır
        sentence[strlen(sentence)] = '\0'; //cumle sinirini belirle

        token = strtok(sentence, AYRAC); //cumledeki ilk kelime alinir
        //cumledeki kelimeler boşluk karakterine gore (" ") parçalanır
        while(token != NULL) {
            strcpy(words[count], token);
            //oneri_count = 0;  //yeni kelime icin oneri count değeri 0'lanır
            words[count][strlen(words[count])] = '\0'; //kelime sinirini belirle

            //case insensitive durum saglanmasi icin kelimedeki butun harfler kucuk harfe cevrilir
            for(j = 0; j <strlen(words[count]); j++) {
                words[count][j] = tolower(words[count][j]);
            }
            count++;    //cümlenin kelimelerini tutan count'u güncelle
            //cümlenin kelimelerini tutan matriste yer genişlet
            words = (char**) realloc(words, (count + 1)*sizeof(char*));
            words[count] = (char*) calloc(WORD_SIZE, sizeof(char));
            token = strtok(NULL, AYRAC); //cumledeki kelimeler alinir
        }
```

Fonksiyondaki işlemler kullanıcı istedikçe dönen bir while döngüsünün içerisinde gerçekleşir. İlk olarak kullanıcıdan girmek istediği cümle alınır ve bu cümle kelime kelime parçalanır, sözlükteki kelimelerle eşleşmesi için harfleri küçük harfe çevrilir ve bu kelimeler bir matriste saklanır.

```
    //for dongusu cumledeki kelime sayısı kadar doner ve her bir kelime icin kontrol gerceklesir
    for(i = 0; i < count; i++) {
        oneri_count = 0;
        isExist = searchHash(sozluk,words[i]); //ilk kelime sözlükte aranir
        //bulunamaması durumunda
        if(isExist == 0) {
            //ilk kelime hatalı kelime sözlüğünde aranır
            isExist_2 = searchHash_2(hatali_sozluk, words[i]);
            //bulunamaması durumunda
            if(isExist_2 == 1000) {
                distances = totalDistances(words[i],sozluk); //Edit Distance'ları hesapla
                //ED'si 1 olan kelimeler var mı ara, varsa oneriler matrisine ekle
                for(j = 0; j < M; j++) {
                    if(distances[j] == 1) {
                        strcpy(oneriler[oneri_count], sozluk[j].word);  //ED'si 1 olan kelimeyi öneri matrisine ekle
                        oneri_count++;  //öneri count'unu güncelle
                        //öneri matrisinin yerini 1 genişlet (sonraki ekleme durumu için)
                        oneriler = (char**) realloc(oneriler, (oneri_count + 1)*sizeof(char*));
                        oneriler[oneri_count] = (char*) calloc(WORD_SIZE, sizeof(char));
                    }
                }
                //ED'si 1 olan kelime yoksa, 2 olanları ara, varsa matrise ekle
                if(oneri_count == 0) {
                    for(j = 0; j < M; j++) {
                        if(distances[j] == 2) {
                            strcpy(oneriler[oneri_count], sozluk[j].word); //ED'si 2 olan kelimeyi öneri matrisine e
                            oneri_count++; //öneri count'unu güncelle
                            //öneri matrisinin yerini 1 genişlet (sonraki ekleme durumu için)
                            oneriler = (char**) realloc(oneriler, (oneri_count + 1)*sizeof(char*));
                            oneriler[oneri_count] = (char*) calloc(WORD_SIZE, sizeof(char));
                        }
                    }
                }
```

Kelimeler matrise alındıktan sonra her bir kelime öncelikle sözlük tablosunda aranır. Bulunursa, kelime olduğu gibi bırakılıp diğer kelimeye geçilir. Bulunamazsa, hatalı sözlük tablosunda aranır. Bulunursa, daha önceden önerilip kullanıcının kabul ettiği kelime kullanıcıya yeniden önerilir. Kelime hatalı kelime sözlüğünde de bulunamazsa, kelimenin sözlükteki diğer tüm kelimelere olan Edit Distance değeri hesaplanır ve dizide saklanır. Sonrasında bu dizide arama yapılır ve ED'si 1 olan kelimeler var mı kontrol edilir. Varsa bunlar öneri matrisine alınır. Yok ise ED'si 2 olan kelimeler var mı kontrol edilir ve varsa bunlar öneri matrisine alınır. ED'si 1 veya 2 olan kelime mevcut değilse kullanıcıya o kelime için öneri yapılmaz. ED'si 1 olan kelimeler var ise 2 olanlar öneri matrisine alınmaz ve kullanıcıya önerilmez.

```
                //öneri matrisinde kelime varsa
                if(oneri_count != 0) {
                    //kullanıcıya hatalı kelimeyi ve önerilebilecek kelimeleri ekrana yazdırarak göster ve seçmesini iste
                    printf("\"%s\" is not in the dictionary. Did you mean: \"%s\"",words[i],oneriler[0]);
                    for(j = 1; j < oneri_count; j++) {
                        printf(" or \"%s\"",oneriler[j]);
                    }
                    printf("\n");
                    scanf("%s",kullanici_onerisi);  //kullanıcının seçimini al
                    printf("");
                    insertTable_2(words[i],kullanici_onerisi,hatali_count,hatali_sozluk);   //hatalı kelimeyi ve kullanıcın seçtiği öneriyi hatalı kelime sözlüğüne
                    strcpy(words[i],kullanici_onerisi); //ekrana yazdırılacak olan son cümlede hatalı kelimeyi önerisi ile değiştir
                }
                //önerilecek kelime bulunamadıysa hatalı kelimeyi olduğu gibi bırak
            }
            //hatalı kelime daha önceden hatalı kelime sözlüğüne eklendiyse, ilk eklenme anında kullanıcının seçtiği öneri yeniden önerilir
            else {
                printf("\"%s\" is not in the dictionary. Did you mean: \"%s\"",words[i],hatali_sozluk[isExist_2].onerilen_kelime);
                printf("\n");
                scanf("%s",kullanici_onerisi);  //kullanıcının seçimini al
                strcpy(words[i],kullanici_onerisi);
                //strcpy(words[i],hatali_sozluk[isExist_2].onerilen_kelime); //daha onceden onerilen kelime ile hatali kelimeyi degistir.
            }
        }
        //kelime hatalı değilse ve sözlükte bulunuyorsa bir şey yapma
    }
```

Uygun ED'li kelimeler öneri matrisinden kullanıcıya önerilir ve kullanıcının seçimi alınır. Kullanıcının seçimi ve hatalı kelime, hatalı kelime sözlüğüne eklenir. Ayrıca alınan hatalı kelime ile kullanıcının seçtiği öneri değiştirilir.

```
255         printf("Final sentence :\n");
256         //cumlenin son haline for dongusu yardımıyla kelime kelime ekrana yazdır
257         words[0][0] = toupper(words[0][0]); //cumledeki ilk harf buyuk harfe cevirilir
258         for(j = 0; j < count; j++) {
259             printf("%s ",words[j]);
260         }
261         printf("\n\nYeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.\n");
262         scanf("%d",&cont);  //kullanıcıdan yeni cumle girisi olup olmayacagının bilgisi alınır
263         clear_input_buffer();   //gets() fonksiyonundan once input bufferı temizle
264     }
265     //dinamik olarak acilan yerler free edilir
266     free(sentence);
267     free(kullanici_onerisi);
268     free(distances);
269     for(i = 0; i < count; i++) {
270         free(words[i]);
271     }
272     free(words);
273     for(i = 0; i < oneri_count; i++) {
274         free(oneriler[i]);
275     }
276     free(oneriler);
277 
278     return ;    //fonksiyonun sonu
279 }
```

while döngüsünün sonunda cümlenin son hali ekrana yazdırılır ve kullanıcıya yeni cümle girişi yapıp yapmayacağı sorulur ve bunun bilgisi alınır. Kullanıcı yeni cümle girişi yapmayacak ise fonksiyonda dinamik olarak açılan yerler free işleminden geçirilir ve fonksiyondan çıkılır.

insertTable ve insertTable_2 Fonksiyonları

```
281 //ilgili kelimeyi alip sozluk icin tanımlanan hash tablosuna ekleyen fonksiyon
282 int insertTable(char *word, int *total_word_count, struct node *hash_table) {
368 
369 //ilgili kelimeyi alip hatali kelime sozlugu icin tanimlanan hash tablosuna ekleyen fonksiyon
370 int insertTable_2(char *word, char *oneri, int *total_word_count, struct node_2 *hash_table) {
457 
```

insertTable fonksiyonu içerisine, sözlük tablosuna eklenecek kelimeyi, sözlük tablosundaki kelime sayısını ve sözlük tablosunu alır. İçerisine aldığı kelimeyi yoksa tabloya ekler ve işlemler tamamlanır. insertTable_2 fonksiyonu ise içerisine hatalı kelime sözlüğü tablosuna eklenecek hatalı kelimeyi ve bu kelimenin önerilip kabul edilen versiyonunu, hatalı kelime sözlüğü tablosundaki kelime sayısını ve hatalı kelime sözlüğü tablosunu alır. İçerisine aldığı kelime tabloda varsa ekleme işlemi yapmaz, kelime yoksa önerisiyle birlikte tabloya ekler.

readInputFile Fonksiyonu

```
458 //dosyadan veriyi satir satir okuyup, kelimeleri sozluk icin tanımlanan hash tablosuna ekleyen fonksiyon
459 int readInputFile(char *file_name, int *total_word_count, struct node *hash_table) {
533 
```

readInputFile fonksiyonu içerisine aldığı isimdeki dosyayı açar ve kelimeleri içerisine aldığı sözlük tablosuna ekler. Böylece sözlük programda bir hash tablosunda oluşturulmuş olur.

searchHash ve searchHash_2 Fonksiyonları

```
534 //icine verilen kelimeyi sozluk icin tanimlanan hash tablosunda arayan fonksiyon / kelime tabloda mevcutsa 1 / değil ise 0 döner
535 int searchHash(struct node *hash_table, char *word) {
578
579 //icine verilen kelimeyi hatali kelime sozlugu icin tanimlanan hash tablosunda arayan fonksiyon / kelime tabloda mevcutsa kelimenin adresi, değil ise 1000 döner
580 int searchHash_2(struct node_2 *hash_table, char *word) {
623
```

searchHash fonksiyonu içerisine aldığı kelimeyi, içerisine aldığı sözlük hash tablosunda arar. Kelime tabloda mevcutsa 1, değilse 0 döner. searchHash_2 fonksiyonu ise içerisine aldığı kelimeyi, içerisine aldığı hatalı kelime sözlüğü tablosunda arar. Kelime tabloda mevcutsa kelimenin tablodaki adresini, mevcut değilse de adres olamayacak bir değer olan 1000 değerini dışarı döndürür.

Ana Fonksiyon

```
625 int main() {
626
627     //FILE *table_file, *table_file_2;  //hashtable'ların okunacağı fileler icin file pointerlar
628     char *file_name = (char*) calloc(TEXT_NAME_SIZE,sizeof(char));  //input dosyasinin adi
629     struct node sozluk[M]; //sozluk icin hash tablosu
630     struct node_2 hatali_kelimeler[M]; //hatali kelime sozlugu icin hash tablosu
631     int total_word_count_1 = 0; //tablodaki kelime sayisini tutan degisken
632     int total_word_count_2 = 0;
633     int kontrol;    //input file'in okunup okunamadiğinin kontrolunu tutan degisken
634     int i;
635
636     //tablolar 0'dan olusturuldugunda ilklendirme foru
637     for(i = 0; i < M; i++) {
638         hatali_kelimeler[i].count = 0;
639         hatali_kelimeler[i].loadFactor = 0;
640         sozluk[i].count = 0;
641         sozluk[i].loadFactor = 0;
642     }
```

Ana fonksiyonda ilk olarak gerekli değişkenler ve diziler tanımlanır. Sonrasında sözlük ve hatalı kelime sözlüğü olacak olan hash tabloları ilklendirilir.

```
//  //sozulugun bulundugu hash tablosunun oldugu dosya acilir
//    if ((table_file = fopen(SOZLUK_TXT, "rb")) == NULL) {
//        printf("Error opening file\n");
//        return 1;
//    }
//    //hash tablosu dosyadan okunur
//    else {
//      fread(sozluk, sizeof(struct node) * M, 1, table_file);  //tabloyu oku
//        fclose(table_file);   //dosyayi kapat
// }
// total_word_count_1 =  sozluk[0].loadFactor * (float) M;
// printf("Sozluk basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablonun doluluk orani (load factor) : %.3f\n\n",M,sozluk[0].loadFactor);
// system("pause");
//-------------------------------------------------------------------------------------------------------------------------------------------------------
// //hatali kelime sozlugunun bulundugu hash tablosunun oldugu dosya acilir
// if ((table_file_2 = fopen(HATALI_TXT, "rb")) == NULL) {
//        printf("Error opening file\n");
//        return 1;
//    }
//    //hash tablosu dosyadan okunur
//    else {
//      fread(hatali_kelimeler, sizeof(struct node_2) * M, 1, table_file_2);    //tabloyu oku
//        fclose(table_file_2); //dosyayi kapat
// }
// total_word_count_2 =  hatali_kelimeler[0].loadFactor * (float) M;
// printf("\nHatali kelime sozlugu basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablonun doluluk orani (load factor) : %.3f\n\n",M,hatali
// system("pause");
```

Ana fonksiyonda hash tablolarının dosyadan okunması ve dosyaya yazılması ihtimaline karşın okuma ve yazma blokları yorum satırlarına alınmıştır.

```
// //hash tablosunun oldugu dosya acilir
//    if ( (table_file = fopen(SOZLUK_TXT, "wb")) == NULL ) {
//        printf("Error opening file\n");
//        return 1;
//    }
//    //sozlugun hash tablosu dosyaya yazilir
// else {
//         fwrite(sozluk, sizeof(struct node) * M, 1, table_file); //tablo dosyaya yazilir
//         fclose(table_file); //dosya kapatilir
//         printf("\nSozluk Dosyaya Basari ile Yazildi!\nTablonun doluluk orani (load factor) : %.3f\n",sozluk[0].loadFactor);
// }
//-------------------------------------------------------------------------------------------------------------------------------------------------------
// if ( (table_file_2 = fopen(HATALI_TXT, "wb")) == NULL ) {
//        printf("Error opening file\n");
//        return 1;
//    }
//    //hatali kelime sozlugunun hash tablosu dosyaya yazilir
// else {
//         fwrite(hatali_kelimeler, sizeof(struct node_2) * M, 1, table_file_2); //tablo dosyaya yazilir
//         fclose(table_file_2);   //dosya kapatilir
//         printf("\nHatali Kelimeler Dosyaya Basari ile Yazildi!\nTablonun doluluk orani (load factor) : %.3f\n",hatali_kelimeler[0].loadFactor);
// }
```

```
    printf("Sozluk dosyasinin adini .txt uzantili olacak sekilde giriniz : ");
    scanf("%s",file_name);
    printf("\n");
    printf("Sozluk dosyadan okunuyor...\n");
    kontrol = readInputFile(file_name,&total_word_count_1,sozluk);  //okuma fonksiyonu cagirilir
    if(kontrol == 1) {
        printf("Input dosyasi okunamadi!\n");
        return 1; //dosya okunamazsa mainden cikilir
    }
    else {
        printf("Sozluk basari ile okundu.\nSozluk tablosunun doluluk orani = %.3f\n\n",sozluk[0].loadFactor);
    }
```

Sözlük dosyadan kelime kelime okunacaksa dosya ismi kullanıcıdan alınır ve readInputFile fonksiyonu yardımıyla txt dosyasındaki kelimeler sözlük hash tablosuna open adressing ve double hashing yöntemleri kullanılarak yerleştirilir.

```
713         clear_input_buffer(); //scanf'den sonra input buffer temizlenir
714         printf("\n");
715         levenshtein(sozluk,hatali_kelimeler,&total_word_count_2);   //Levenshtein fonksiyonu çağırılarak cümleler alınmaya ve işlenmeye başlanır
716
717         free(file_name); //free islemi
718         return 0;   //end of main
719     }
```

Ana fonksiyonun son kısmında levenshtein fonksiyonu çağırılarak işlemler başlatılır ve fonksiyondan çıkıldığı durumda free işleminin ardından program sonlandırılır.

PROGRAM ÇIKTILARI

```
E:\YTU\BLM3-GUZ\alg analizi\odev-3\17011501_2.exe
Sozluk dosyasinin adini .txt uzantili olacak sekilde giriniz : smallDictionary.txt

Sozluk dosyadan okunuyor...

Sozluk basari ile okundu.
Sozluk tablosunun doluluk orani = 0.339


Cumleyi giriniz...
It is coold
"coold" is not in the dictionary. Did you mean: "hold" or "good"
good
Final sentence :
It is good

Yeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.
1
Cumleyi giriniz...
It is coold
"coold" is not in the dictionary. Did you mean: "good"
good
Final sentence :
It is good

Yeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.
```

```
Yeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.
1
Cumleyi giriniz...
ths is best algoritm for search
"ths" is not in the dictionary. Did you mean: "the" or "th" or "this"
this
"best" is not in the dictionary. Did you mean: "be" or "next" or "but" or "last" or "get" or "less" or "must" or "list"
or "most" or "been"
next
"algoritm" is not in the dictionary. Did you mean: "algorithm"
algorithm
"search" is not in the dictionary. Did you mean: "each"
each
Final sentence :
This is next algorithm for each

Yeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.
```

SOURCE CODE

```
1.  /*
2.  @file
3.  BLM3021 2020-2021 GUZ ODEV-3 / Soru-2
4.  Bu programda sorgulanan bir cümlede, yanlış yazılmış kelimelerin yerine doğru kelime
    ler öneren bir sistem tasarımı gerçeklenmiştir
5.
6.  @author
7.  İsim: Ahmet Said SAĞLAM
8.  Öğrenci No: 17011501
9.  Tarih: 19.12.2020
10. E-Mail: l1117501@std.yildiz.edu.tr
11. Compiler: TDM-GCC 4.9.2 64 bit-Release
12. IDE: DEV-C++ (version 5.11)
13. İşletim Sistemi: Windows 10 Pro 64 bit
14. */
15.
16. #include <stdio.h>
17. #include <stdlib.h>
18. #include <string.h>
19. #include <conio.h>
20. #include <stdbool.h>
21. #include <ctype.h>
22.
23. #define SENTENCE_SIZE 200   //kullanıcıdan alınacak olan maksimum cümle uzunlugu
24. #define BUFFER_SIZE 10000 //txt dosyadan alınan satırın saklanacağı bufferin boyutu

25. #define TEXT_NAME_SIZE 20 // max file ismi uzunlugu
26. #define AYRAC " "   //kelimelerin ayrilacagi delim ifadesi
27. #define HORNER_NUMBER 7 //horner numarasi hesaplanirken kullanilcak olan asal sayi
28. #define WORD_SIZE 20    //alinan bir kelimenin maksimum uzunlugu
29. #define MM 996 //double hashing hesabinda, iki numarali keyi hesaplarken kullanilaca
    k olan sayi
30. #define M 997 //hash tablosunun boyutu
31. #define SOZLUK_TXT "sozluk.txt" //sozlugu tutan hash tablosunun kaydedildigi dosyani
    n ismi
32. #define HATALI_TXT "hatalikelime.txt"   //hatali kelime sozlugunu tutan hash tablosu
    nun kaydedildigi dosyanin ismi
33. #define K 2 //Maksimum olabilecek Edit Distance
34.
35. //sozlukte tutulacak kelimeler icin structure
36. typedef struct node {
37.     int count;  //goz bos ise 0, dolu ise 1 degerini alir
38.     float loadFactor;   //tablonun doluluk orani
39.     char word[WORD_SIZE];   //tutulan kelime
40. } node;
41.
42. //hatali kelime sozlugunde tutulacak olan elemanlar icin structure
43. typedef struct node_2 {
44.     int count;  //goz bos ise 0, dolu ise 1 degerini alır
45.     float loadFactor;   //tablonun doluluk orani
46.     char word[WORD_SIZE];   //tutulan hatalı kelime
47.     char onerilen_kelime[WORD_SIZE]; //önerilen kelime
48. } node_2;
49.
50. //gets() fonksiyonu kullanılmadan önce eger scanf kullanıldıysa gets() duzgun calism
    ayacagi icin input buffer'i temizlemmek icin gerekli fonksiyon
51. int clear_input_buffer(void) {
52.     int ch;
53.     while (((ch = getchar()) != EOF) && (ch != '\n'));
54.     return ch;
55. }
```

```
56. 
57. //icine verilen kelimenin horner sayisini donduren fonksiyon
58. long long getHorner(char *word) {
59.     long long key = 0L; //kelimenin sayisal key karşılığını tutacak olan degisken
60.     int i; //dongu degiskeni
61.     //kelimenin horner sayisi hesaplanir
62.     for(i = 0; i < strlen(word); i++) {
63.         key = (long long) (HORNER_NUMBER * key + (word[i] - 'a' + 1));
64.     }
65.     return (long long) key;
66. 
67. }
68. 
69. //icine verilen sayilardan kucuk olanı donduren fonksiyon
70. int min(int x, int y) {
71.     if(x < y) {
72.         return x;
73.     }
74.     return y;
75. }
76. 
77. //icine aldıgı 3 sayıdan en kucugunu donduren fonksiyon
78. int min_3(int x, int y, int z) {
79.     return min(min(x,y), z);
80. }
81. 
82. //icine aldıgı iki karakter farklıysa 1, aynıysa 0 donduren fonksiyon
83. int diff(char x, char y) {
84.     if(x == y) {
85.         return 0;
86.     }
87.     return 1;
88. }
89. 
90. //icerisine aldıgı iki kelimenin L.Edit Distance'nı hesaplayan fonksiyon
91. int calculateDistance(char *word_1, char *word_2, int w1_length, int w2_length) {
92.     //Edit Distance Kesin Olarak 2'den büyükse(kelimelerin uzunlukları farkı 2'den b
    üyükse kesin olarak 2'den büyüktür) hesaplama yapmadan fonksiyondan çıkılır.
93.     if(w1_length - w2_length > K || w2_length - w1_length > K) {
94.         return 99;  //distance olarak 99 dondurulur. Anlamsız ve kullanılmayacak bir
     degerdir
95.     }
96.     int ED[w1_length + 1][w2_length + 1]; //Edit Distance hesabında kullanılan matri
    s olusturulur
97.     int i, j; //dongu degiskenleri
98.     int control = 0;    //ED kesin olarak 2'den buyuk mu diye kontrolun yapılacagı d
    egisken
99. 
100.            i = 0;  //i ilklendirilir
101.            //i satır sayısı kadar donsun ve control sutun sayisindan kucuk ise donsu
    n
102.            while(i <= w1_length && control <= w2_length) {
103.                control = 0;    //her satir icin control 0'lanır
104.                //satırda sutun sayısı kadar dön matrisin gözlerine eris
105.                for(j = 0; j <= w2_length; j++) {
106.                    //matrisin 0. satırını ilklendir
107.                    if(i == 0) {
108.                        ED[i][j] = j;
109.                    }
110.                    //matrisin 0. sutunu ilklendirilir
111.                    else if(j == 0) {
112.                        ED[i][j] = i;
113.                    }
114.                    //kelimelerin ilgili harfinde insert-delete-
    replace işlemlerinden en az mesafe hangisi ise o seçilir ve matris doldurulur.
115.                    else {
```

```
                    ED[i][j] = min_3((ED[i-1][j] + 1), (ED[i][j-1] + 1), (ED[i-1][j-1] + diff(word_1[i-1],word_2[j-1])));
                }
                //matrisin ilgili satirindaki ilgili gözün değeri 2'den büyükse control degiskenini 1 arttır
                if(ED[i][j] > K) {
                    control++;
                }
                //printf("kontrol - control : %d------satir : %d -- sutun : %d\n",control,i,j);
            }
            i++;    //yeni satıra gecmek icin i'yi guncelle
        }

        //while dongusunden matrisin sonuna gelindigi icin cıkıldıysa, Edit Distance degeri disari dondurulur
        if(control <= w2_length) {
            return ED[w1_length][w2_length];    //Edit Distance dışarı dondurulur

        }
        //while dongusunden, control sutun sayısına esit oldugu icin cikildiysa, matrisin o satırındaki tum degerler 2'den buyuk demektir
        //bir alt satıra inecek olan tum degerler 2'den buyuk olacagı icin Edit Distance'ın 2'den buyuk olacağı kesinlesmistir.
        else {
            return 99; //distance olarak 99 dondurulur. Anlamsız ve kullanılmayacak bir degerdir
        }
    }

    //içerisine aldığı kelime ile sözlükteki kelimelerin mesafelerini hesaplar ve dizide kaydeder. Diziyi dışarı döndürür.
    int *totalDistances(char *word, struct node *sozluk) {
        int *distances = (int*) malloc(M * sizeof(int));    //sözlükteki kelimelerin, fonksiyonun içine aldığı kelimeye göre ED'sini tutan dizi
        int i;  //dongu degiskeni
        for (i = 0; i < M; i++) {
            //hash tablosunda(sozlukte) ilgili göz boş ise
            if(sozluk[i].count == 0) {
                distances[i] = 99;  //distance olarak 99 atanır. Anlamsız bir değerdir.
            }
            else {
                distances[i] = calculateDistance(word, sozluk[i].word, strlen(word), strlen(sozluk[i].word)); //ilgili ED hesaplanır
            }
        }
        return distances;//dizi dışarı döndürülür
    }

    //cumleyi kullanıcıdan alarak gerektiginde oneri yapan ve cumlenin son halini ekrana yazdıran fonksiyon
    void levenshtein(struct node *sozluk, struct node_2 *hatali_sozluk, int *hatali_count) {
        int count = 0;  //cumlede kac adet kelime var tutan degisken
        int oneri_count = 0;    //onerilebilecek kelime sayısı
        char *sentence = (char*) calloc(SENTENCE_SIZE, sizeof(char));   //cumle için yer açılır
        char **words = (char**) calloc(1, sizeof(char*));   //cumleden parcalanan kelimeleri tutmak icin matris
        words[count] = (char*) calloc(WORD_SIZE, sizeof(char)); //matrisin her satırında kelime tutmak için yer açılır
        char **oneriler = (char**) calloc(1, sizeof(char*)); //ED'si 1 veya 2 olan onerilecek kelimeleri tutmak icin matris
        oneriler[oneri_count] = (char*) calloc(WORD_SIZE, sizeof(char));    //matrisin her satırında kelime tutmak için yer açılır
```

```
163.          char *kullanici_onerisi = (char*) calloc(WORD_SIZE, sizeof(char));  //kul
    lanıcı önerilen kelimeler arasından seçim yapar. Bu seçimi tutan dizi için yer açılı
    r
164.          int cont = 1; //yeni cumle girisini kontrol eden degisken
165.          int isExist; //kelime sozluk icin tanimlanan hash table'da var mi yok mu
    donus degerini tutan degisken(kelime tabloda mevcutsa 1, degilse 0)
166.          int isExist_2; //kelime hatali kelime sozlgu icin tanimlanan hash table'd
    a var mi yok mu donus degerini tutan degisken(kelime tabloda mevcutsa adresi, yoksa
    1000)
167.          int j,i;    //dongu degiskenleri
168.          int *distances; //Edit distance'ları tutan dizi için tanımlama
169.          char *token;    //strtok ile alinan kelime
170.
171.          //kullanıcı istedikçe dönen while
172.          while(cont) {
173.              count = 0;  //her yeni cümlede yeni aramalar icin count değerleri 0'l
    anır
174.              oneri_count = 0;    //her yeni cümlede yeni aramalar icin count değer
    leri 0'lanır
175.              printf("Cumleyi giriniz...\n");
176.              gets(sentence); //cümle kullanıcıdan alınır
177.              sentence[strlen(sentence)] = '\0'; //cumle sinirini belirle
178.
179.              token = strtok(sentence, AYRAC); //cumledeki ilk kelime alinir
180.              //cumledeki kelimeler boşluk karakterine gore (" ") parçalanır
181.              while(token != NULL) {
182.                  strcpy(words[count], token);
183.                  //oneri_count = 0;  //yeni kelime icin oneri count değeri 0'lanır

184.                  words[count][strlen(words[count])] = '\0'; //kelime sinirini beli
    rle
185.
186.                  //case insensitive durum saglanmasi icin kelimedeki butun harfler
     kucuk harfe cevrilir
187.                  for(j = 0; j <strlen(words[count]); j++) {
188.                      words[count][j] = tolower(words[count][j]);
189.                  }
190.                  count++;    //cümlenin kelimelerini tutan count'u güncelle
191.                  //cümlenin kelimelerini tutan matriste yer genişlet
192.                  words = (char**) realloc(words, (count + 1)*sizeof(char*));
193.                  words[count] = (char*) calloc(WORD_SIZE, sizeof(char));
194.                  token = strtok(NULL, AYRAC); //cumledeki kelimeler alinir
195.              }
196.              //for dongusu cumledeki kelime sayısı kadar doner ve her bir kelime i
    cin kontrol gerceklesir
197.              for(i = 0; i < count; i++) {
198.                  oneri_count = 0;
199.                  isExist = searchHash(sozluk,words[i]); //ilk kelime sözlükte aran
    ir
200.                  //bulunamaması durumunda
201.                  if(isExist == 0) {
202.                      //ilk kelime hatalı kelime sözlüğünde aranır
203.                      isExist_2 = searchHash_2(hatali_sozluk, words[i]);
204.                      //bulunamaması durumunda
205.                      if(isExist_2 == 1000) {
206.                          distances = totalDistances(words[i],sozluk); //Edit Dista
    nce'ları hesapla
207.                          //ED'si 1 olan kelimeler var mı ara, varsa oneriler matri
    sine ekle
208.                          for(j = 0; j < M; j++) {
209.                              if(distances[j] == 1) {
210.                                  strcpy(oneriler[oneri_count], sozluk[j].word);  /
    /ED'si 1 olan kelimeyi öneri matrisine ekle
211.                                  oneri_count++;  //öneri count'unu güncelle
212.                                  //öneri matrisinin yerini 1 genişlet (sonraki ekl
    eme durumu için)
```

```
213.                                     oneriler = (char**) realloc(oneriler, (oneri_coun
    t + 1)*sizeof(char*));
214.                                     oneriler[oneri_count] = (char*) calloc(WORD_SIZE,
     sizeof(char));
215.                                 }
216.                             }
217.                             //ED'si 1 olan kelime yoksa, 2 olanlari ara, varsa matris
    e ekle
218.                             if(oneri_count == 0) {
219.                                 for(j = 0; j < M; j++) {
220.                                     if(distances[j] == 2) {
221.                                         strcpy(oneriler[oneri_count], sozluk[j].word)
    ; //ED'si 2 olan kelimeyi öneri matrisine ekle
222.                                         oneri_count++; //öneri count'unu güncelle
223.                                         //öneri matrisinin yerini 1 genişlet (sonraki
     ekleme durumu için)
224.                                         oneriler = (char**) realloc(oneriler, (oneri_
    count + 1)*sizeof(char*));
225.                                         oneriler[oneri_count] = (char*) calloc(WORD_S
    IZE, sizeof(char));
226.                                     }
227.                                 }
228.                             }
229.                             //öneri matrisinde kelime varsa
230.                             if(oneri_count != 0) {
231.                                 //kullanıcıya hatalı kelimeyi ve önerilebilecek kelim
    eleri ekrana yazdırarak göster ve seçmesini iste
232.                                 printf("\"%s\" is not in the dictionary. Did you mean
    : \"%s\"",words[i],oneriler[0]);
233.                                 for(j = 1; j < oneri_count; j++) {
234.                                     printf(" or \"%s\"",oneriler[j]);
235.                                 }
236.                                 printf("\n");
237.                                 scanf("%s",kullanici_onerisi);  //kullanıcının seçimi
    ni al
238.                                 printf("");
239.                                 insertTable_2(words[i],kullanici_onerisi,hatali_count
    ,hatali_sozluk);   //hatalı kelimeyi ve kullanıcın seçtiği öneriyi hatalı kelime söz
    lüğüne ekle
240.                                 strcpy(words[i],kullanici_onerisi); //ekrana yazdırıl
    acak olan son cümlede hatalı kelimeyi önerisi ile değiştir
241.                             }
242.                             //önerilecek kelime bulunamadıysa hatalı kelimeyi oldugu
    gibi bırak
243.                         }
244.                         //hatalı kelime daha önceden hatalı kelime sözlüğüne eklendiy
    se, ilk eklenme anında kullanıcının seçtiği öneri yeniden önerilir
245.                         else {
246.                             printf("\"%s\" is not in the dictionary. Did you mean: \"
    %s\"",words[i],hatali_sozluk[isExist_2].onerilen_kelime);
247.                             printf("\n");
248.                             scanf("%s",kullanici_onerisi);  //kullanıcının seçimini a
    l
249.                             strcpy(words[i],kullanici_onerisi);
250.                             //strcpy(words[i],hatali_sozluk[isExist_2].onerilen_kelim
    e); //daha onceden onerilen kelime ile hatali kelimeyi degistir.
251.                         }
252.                     }
253.                     //kelime hatalı değilse ve sözlükte bulunuyorsa bir şey yapma
254.                 }
255.                 printf("Final sentence :\n");
256.                 //cumlenin son haline for dongusu yardımıyla kelime kelime ekrana yaz
    dır
257.                 words[0][0] = toupper(words[0][0]); //cumledeki ilk harf buyuk harfe
    cevirilir
258.                 for(j = 0; j < count; j++) {
```

```
259.                 printf("%s ",words[j]);
260.             }
261.             printf("\n\nYeni cumle girisi icin 1'e, cikmak icin 0'a basiniz.\n");

262.             scanf("%d",&cont);  //kullanıcıdan yeni cumle girisi olup olmayacagın
    ın bilgisi alınır
263.             clear_input_buffer();   //gets() fonksiyonundan once input bufferı te
    mizle
264.         }
265.         //dinamik olarak acilan yerler free edilir
266.         free(sentence);
267.         free(kullanici_onerisi);
268.         free(distances);
269.         for(i = 0; i < count; i++) {
270.             free(words[i]);
271.         }
272.         free(words);
273.         for(i = 0; i < oneri_count; i++) {
274.             free(oneriler[i]);
275.         }
276.         free(oneriler);
277.
278.         return ;    //fonksiyonun sonu
279.     }
280.
281.     //ilgili kelimeyi alip sozluk icin tanımlanan hash tablosuna ekleyen fonksiyo
    n
282.     int insertTable(char *word, int *total_word_count, struct node *hash_table) {

283.         int i = 0; //adres hesabinda adim sayisi
284.         long long key = getHorner(word);    //kelimenin horner sayisi
285.         int adr;    //kelimenin tablodaki adresi
286.
287.         //adres double probing yontemi ile hesaplanir
288.         int h1_key = key % M;
289.         int h2_key = 1 + (key % MM);
290.         adr = (h1_key + (i * h2_key)) % M;
291.         i++;    //adim sayisi olasi bir yeni hesaplama icin guncellenir
292.
293.         //printf("\n\n %s , adres: %d\n\n",word,adr);
294.         //system("pause");
295.
296.         //load factor 0.8'in ustundeyse uyari verilir
297.         if(hash_table[0].loadFactor > 0.8 && hash_table[0].loadFactor < 1) {
298.             printf("UYARI. Load Factor sinirina yaklasiyorsunuz!\nLoad Factor : %
    .2f\n",hash_table[0].loadFactor);
299.         }
300.         //tablo dolduysa uyari verilir
301.         if(hash_table[0].loadFactor >= 1) {
302.             printf("UYARI. Load Factor sinirini gectiniz!\nLoad Factor : %.2f\n",
    hash_table[0].loadFactor);
303.
304.             //eger kelime daha onceden tabloda var mi diye tablo gezilir
305.             while(i <= M && strcmp(hash_table[adr].word, word) != 0) {
306.                 adr = (h1_key + (i * h2_key)) % M;
307.                 i++;
308.             }
309.             //kelime tabloda mevcutsa
310.             if(strcmp(hash_table[adr].word, word) == 0) {
311.                 //printf("%s kelimesi tabloda mevcut\n",word);
312.                 return -1;
313.             }
314.             //kelime tabloda mevcut degilse, tabloya eklenemedigi uyarisi verilir

315.             else {
```

```
316.                    //printf("%s kelimesi tabloya eklenemedi\n",word); //bilgilendirm
    e printi
317.                    return 2; //fonksiyondan cikilir
318.                }
319.            }
320.            //kelimenin adresi ilk aramada bossa kelime eklenmemis demektir
321.            if(hash_table[adr].count == 0) {
322.                struct node newnode; //yeni bir node tanimlanir
323.                strcpy(newnode.word, word); //kelime yeni node atanir
324.
325.                //printf("\n\n %s , adres: %d\n\n",word,adr);
326.                //system("pause");
327.
328.                newnode.count = 1;  //node'un file countu guncellenir
329.
330.                hash_table[adr] = newnode;  //tabloda ilgili adres node'a esitlenir
331.                *total_word_count = *total_word_count + 1;  //toplam kelime sayisi ar
    ttirilir
332.                hash_table[0].loadFactor = (float) *total_word_count / (float) M; //y
    eni load factor hesaplanip tabloda saklanir
333.                //printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].word)
    ;  //bilgilendirme printleri yazilir
334.                //printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
335.
336.                return 1;   //fonksiyondan cikilir
337.            }
338.            else {
339.                //ilk aramada hash tablosunun gözü doluysa kelime kontrolü yapılarak
    ve gerekirse yeni adres hesaplanarak ilerlenir
340.                while(hash_table[adr].count != 0 && strcmp(hash_table[adr].word, word
    ) != 0) {
341.                    adr = (h1_key + (i * h2_key)) % M;
342.                    i++;
343.                }
344.                //whiledan kelime ile karsilasildigi icin cikildiysa, dosya ismi ekle
    nmek uzere if kosuluna girilir
345.                if(strcmp(hash_table[adr].word, word) == 0) {
346.
347.                    //printf("%s kelimesi tabloda mevcut\n",word);
348.                    return -1; //fonksiyondan cikilir
349.                }
350.                //whiledan bos adrese gelindigi icin cikildiysa
351.                else {
352.                    //yeni bir node tanimlanir ve veriler node kopyalanir
353.                    struct node newnode;
354.                    strcpy(newnode.word, word);
355.                    newnode.count = 1;
356.
357.                    hash_table[adr] = newnode; //olusturulan node tabloda ilgili adre
    se atilir
358.                    *total_word_count = *total_word_count + 1; //tablodaki kelime say
    isi guncellenir
359.                    hash_table[0].loadFactor =(float) *total_word_count / (float) M;
    //load factor hesaplanir
360.                    //printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].w
    ord); //bilgilendirme printleri atilir
361.                    //printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
362.                    //printf("count : %d\n",*total_word_count);
363.
364.                    return 1; //fonksiyondan cikilir
365.                }
366.            }
367.        }
368.
369.        //ilgili kelimeyi alip hatali kelime sozlugu icin tanimlanan hash tablosuna e
    kleyen fonksiyon
```

```
370.       int insertTable_2(char *word, char *oneri, int *total_word_count, struct node
    _2 *hash_table) {
371.           int i = 0; //adres hesabinda adim sayisi
372.           long long key = getHorner(word);    //kelimenin horner sayisi
373.           int adr;    //kelimenin tablodaki adresi
374.
375.           //adres double probing yontemi ile hesaplanir
376.           int h1_key = key % M;
377.           int h2_key = 1 + (key % MM);
378.           adr = (h1_key + (i * h2_key)) % M;
379.           i++;    //adim sayisi olasi bir yeni hesaplama icin guncellenir
380.
381.           //printf("\n\n %s , adres: %d\n\n",word,adr);
382.           //system("pause");
383.
384.           //load factor 0.8'in ustundeyse uyari verilir
385.           if(hash_table[0].loadFactor > 0.8 && hash_table[0].loadFactor < 1) {
386.               printf("UYARI. Load Factor sinirina yaklasiyorsunuz!\nLoad Factor : %
    .2f\n",hash_table[0].loadFactor);
387.           }
388.           //tablo dolduysa uyari verilir
389.           if(hash_table[0].loadFactor >= 1) {
390.               printf("UYARI. Load Factor sinirini gectiniz!\nLoad Factor : %.2f\n",
    hash_table[0].loadFactor);
391.
392.               //eger kelime daha onceden tabloda var mi diye tablo gezilir
393.               while(i <= M && strcmp(hash_table[adr].word, word) != 0) {
394.                   adr = (h1_key + (i * h2_key)) % M;
395.                   i++;
396.               }
397.               //kelime tabloda mevcutsa
398.               if(strcmp(hash_table[adr].word, word) == 0) {
399.                   //printf("%s kelimesi tabloda mevcut\n",word);
400.                   return -1;
401.               }
402.               //kelime tabloda mevcut degilse, tabloya eklenemedigi uyarisi verilir

403.               else {
404.                   //printf("%s kelimesi tabloya eklenemedi\n",word); //bilgilendirm
    e printi
405.                   return 2; //fonksiyondan cikilir
406.               }
407.           }
408.           //kelimenin adresi ilk aramada bossa kelime eklenmemis demektir
409.           if(hash_table[adr].count == 0) {
410.               struct node_2 newnode; //yeni bir node tanimlanir
411.               strcpy(newnode.word, word); //kelime yeni node atanir
412.               strcpy(newnode.onerilen_kelime, oneri);
413.               //printf("\n\n %s , adres: %d\n\n",word,adr);
414.               //system("pause");
415.
416.               newnode.count = 1;  //node'un file countu guncellenir
417.
418.               hash_table[adr] = newnode;  //tabloda ilgili adres node'a esitlenir
419.               *total_word_count = *total_word_count + 1;  //toplam kelime sayisi ar
    ttirilir
420.               hash_table[0].loadFactor = (float) *total_word_count / (float) M; //y
    eni load factor hesaplanip tabloda saklanir
421.               //printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].word)
    ;  //bilgilendirme printleri yazilir
422.               //printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
423.
424.               return 1;   //fonksiyondan cikilir
425.           }
426.           else {
```

```
427.                //ilk aramada hash tablosunun gözü doluysa kelime kontrolü yapılarak
    ve gerekirse yeni adres hesaplanarak ilerlenir
428.                while(hash_table[adr].count != 0 && strcmp(hash_table[adr].word, word
    ) != 0) {
429.                    adr = (h1_key + (i * h2_key)) % M;
430.                    i++;
431.                }
432.                //whiledan kelime ile karsilasildigi icin cikildiysa, dosya ismi ekle
    nmek uzere if kosuluna girilir
433.                if(strcmp(hash_table[adr].word, word) == 0) {
434.
435.                    //printf("%s kelimesi tabloda mevcut\n",word);
436.                    return -1; //fonksiyondan cikilir
437.                }
438.                //whiledan bos adrese gelindigi icin cikildiysa
439.                else {
440.                    //yeni bir node tanimlanir ve veriler node kopyalanir
441.                    struct node_2 newnode;
442.                    strcpy(newnode.word, word);
443.                    strcpy(newnode.onerilen_kelime, oneri);
444.                    newnode.count = 1;
445.
446.                    hash_table[adr] = newnode; //olusturulan node tabloda ilgili adre
    se atilir
447.                    *total_word_count = *total_word_count + 1; //tablodaki kelime say
    isi guncellenir
448.                    hash_table[0].loadFactor =(float) *total_word_count / (float) M;
    //load factor hesaplanir
449.                    //printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].w
    ord); //bilgilendirme printleri atilir
450.                    //printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
451.                    //printf("count : %d\n",*total_word_count);
452.
453.                    return 1; //fonksiyondan cikilir
454.                }
455.            }
456.        }
457.
458.        //dosyadan veriyi satir satir okuyup, kelimeleri sozluk icin tanımlanan hash
    tablosuna ekleyen fonksiyon
459.        int readInputFile(char *file_name, int *total_word_count, struct node *hash_t
    able) {
460.            FILE *inputFile; //input file 'ı açmaya yarayan file pointer
461.            char ch; //dosyadan karakterler okunup bu degiskene aktarilir(satir sayis
    ini hesaplamak icin)
462.            char *buffer = (char*) calloc(BUFFER_SIZE,sizeof(char)); //dosyadan alina
    n satirin tutuldugu buffer
463.            char *word  = (char*) calloc (WORD_SIZE,sizeof(char));//satirdan parcalan
    ip alinan kelime
464.            char *org_word = (char*) calloc (WORD_SIZE,sizeof(char));//orjinal kelime
    yi tutar
465.            int line_count = 0; //dosyadaki satir sayisini tutan degisken
466.            int i, j; //dongu degiskeni
467.            int isExist; //kelime hash table'da var mi yok mu donus degerini tutan de
    gisken
468.            char *token;    //dosyadan okunan kelimeler strtok() yardımıyla parcalanı
    p gecici olarak bu degiskende tutulur
469.            //int wordCount = total_word_count;
470.
471.            if((inputFile = fopen(file_name,"r")) == NULL) {
472.                printf("Dosya okunmak icin acilamadi!\n");
473.                return 1;
474.            }
475.            else {
476.                //dosyadaki satir sayisini hesaplayan do-while bloğu
477.                do
```

```
478.                {
479.                    ch = fgetc(inputFile); //karakter oku
480.                    //new line ise line_count'u 1 arttır
481.                    if (ch == '\n') {
482.                        line_count++;
483.                    }
484.                } while (ch != EOF); //dosya sonuna gelene kadar
485.                rewind(inputFile); //dosyada basa don
486.                line_count++; //line count son haline guncellenir
487.
488.                //dosyadan veriler satir satir okunur ve isleme alinir
489.                for(i = 0; i < line_count; i++) {
490.                    fgets(buffer,BUFFER_SIZE * sizeof(char),inputFile); //satırı dosy
    adan buffer'a al
491.                    buffer[strlen(buffer)] = '\0'; //bufferin sinirini belirle
492.                    //printf("-----------------------------------------------------
    \n");
493.                    //printf("Okunan satir : %s\n\n",buffer);
494.                    //system("PAUSE");
495.                    printf("\n");
496.
497.                    token = strtok(buffer, AYRAC);  //dosyadaki ilk kelime alınır
498.
499.                    while( token != NULL ) {
500.                        strcpy(word, token);
501.                        //satir sonuna gelindiyse new line karakter maskelenir
502.                        if(word[strlen(word)-1] == '\n') {
503.                            word[strlen(word)-1] = '\0';
504.                        }
505.                        strcpy(org_word,word); //alinan orjinal kelime org_word'de sa
    klanir
506.                        //case insensitive durum saglanmasi icin kelimedeki butun har
    fler kucuk harfe cevrilir
507.                        for(j = 0; j <strlen(word); j++) {
508.                            word[j] = tolower(word[j]);
509.                        }
510.                        word[strlen(word)] = '\0';  //kelime siniri belirlenir
511.       //               printf("word : %s, size %d\n",word,strlen(word));
512.       //               system("PAUSE");
513.                        isExist = insertTable(word,total_word_count,hash_table); //ke
    lime hash tablosuna eklenmek uzere insertTable fonksiyonu cagirilir
514.                        //insert table donus degerleri kontrol edilip bilgilendirme p
    rintleri atılır.
515.       //               if(isExist == -1) {
516.       //                   printf("%s kelimesi zaten tabloda mevcut!\n\n",org_word);
     //kelime hash tablosunda mevcutsa blgilendirme printi atilir
517.       //               }
518.       //               else if(isExist == 1) {
519.       //                   //printf("%s kelimesi eklendi!\n\n",org_word);
520.       //               }
521.                        token = strtok(NULL, AYRAC); //satirdaki sonraki kelime alini
    r
522.                    }
523.
524.                }
525.                fclose(inputFile); //dosya kapatilir
526.            }
527.            //free islemleri
528.            free(buffer);
529.            free(word);
530.            free(org_word);
531.            return 0; //fonksiyondan cikilir
532.        }
533.
534.        //icine verilen kelimeyi sozluk icin tanimlanan hash tablosunda arayan fonksi
    yon / kelime tabloda mevcutsa 1 / değil ise 0 döner
```

```
535.       int searchHash(struct node *hash_table, char *word) {
536.           int i = 0; //kelimenin tablodaki adresi double probinge gore hesaplanirke
    n adim sayisini tutan degisken
537.           long long key; //kelimenin horner methoduna gore key degeri
538.           int adr; //kelimenin tablodaki adresi
539.           int j;  //dongu degiskeni
540.
541.           word[strlen(word)] = '\0';  //kelime siniri belirlenir
542.           //kelime case insensitive'lik saglanmasi icin kucuk harflere cevirilir
543.       //  for(j = 0; j <strlen(word); j++) {
544.       //          word[j] = tolower(word[j]);
545.       //  }
546.           key = (long long) getHorner(word);  //horner sayisi alinir
547.
548.           //kelimenin tablodaki adresi double probing yontemiyle hesaplanir
549.           int h1_key = key % M;
550.           int h2_key = 1 + (key % MM);
551.           adr = (h1_key + (i * h2_key)) % M;
552.           i++;  //adim sayisi guncellenir
553.           //kelime tabloda mevcut degilse ilgili gozde bulunan struct'ın text_count
     degeri 0 demektir.
554.           if(hash_table[adr].count == 0) {
555.               //printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda tamaml
    anmistir.\n\n",i);
556.               return 0;
557.           }
558.           //kelimenin tabloda mevcut olma ihtimalinde
559.           else {
560.               //kelime ile karsilasana kadar veya bos goz gorene kadar donen while

561.               while(hash_table[adr].count != 0 && strcmp(hash_table[adr].word, word
    ) != 0) {
562.                   adr = (h1_key + (i * h2_key)) % M;  //adres degeri her adımda gun
    cellenir
563.                   i++;    //adim sayisi guncellenir
564.               }
565.               //kelime ile karsilasildigi  icin while'dan cikilmissa kelime tabloda
     mevcuttur
566.               if(strcmp(hash_table[adr].word, word) == 0) {
567.       //          printf("Kelime tabloda mevcut!\n");
568.       //          printf("Arama islemi %d adimda tamamlanmistir.\n",i);
569.                   return 1;
570.               }
571.               //bos goze gelindigi icin whiledan cikilmissa kelime tabloda mevcut d
    egildir
572.               else {
573.                   //printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda ta
    mamlanmistir.\n",i);
574.                   return 0;
575.               }
576.           }
577.       }
578.
579.       //icine verilen kelimeyi hatali kelime sozlugu icin tanimlanan hash tablosund
    a arayan fonksiyon / kelime tabloda mevcutsa kelimenin adresi, değil ise 1000 döner

580.       int searchHash_2(struct node_2 *hash_table, char *word) {
581.           int i = 0; //kelimenin tablodaki adresi double probinge gore hesaplanirke
    n adim sayisini tutan degisken
582.           long long key; //kelimenin horner methoduna gore key degeri
583.           int adr; //kelimenin tablodaki adresi
584.           int j;  //dongu degiskeni
585.
586.           word[strlen(word)] = '\0';  //kelime siniri belirlenir
587.           //kelime case insensitive'lik saglanmasi icin kucuk harflere cevirilir
588.       //  for(j = 0; j <strlen(word); j++) {
```

```
589.        //          word[j] = tolower(word[j]);
590.        //  }
591.            key = (long long) getHorner(word);  //horner sayisi alinir
592.
593.            //kelimenin tablodaki adresi double probing yontemiyle hesaplanir
594.            int h1_key = key % M;
595.            int h2_key = 1 + (key % MM);
596.            adr = (h1_key + (i * h2_key)) % M;
597.            i++;  //adim sayisi guncellenir
598.            //kelime tabloda mevcut degilse ilgili gozde bulunan struct'ın text_count
     degeri 0 demektir.
599.            if(hash_table[adr].count == 0) {
600.                //printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda tamaml
    anmistir.\n\n",i);
601.                return 1000;    //adres olamayacak bir deger dondurulur
602.            }
603.            //kelimenin tabloda mevcut olma ihtimalinde
604.            else {
605.                //kelime ile karsilasana kadar veya bos goz gorene kadar donen while

606.                while(hash_table[adr].count != 0 && strcmp(hash_table[adr].word, word
    ) != 0) {
607.                    adr = (h1_key + (i * h2_key)) % M;  //adres degeri her adımda gun
    cellenir
608.                    i++;    //adim sayisi guncellenir
609.                }
610.                //kelime ile karsilasildigi  icin while'dan cikilmissa kelime tabloda
     mevcuttur
611.                if(strcmp(hash_table[adr].word, word) == 0) {
612.        //          printf("Kelime tabloda mevcut!\n");
613.        //          printf("Arama islemi %d adimda tamamlanmistir.\n",i);
614.                    return adr; //bulunan kelimenin adresi dondurulur
615.                }
616.                //bos goze gelindigi icin whiledan cikilmissa kelime tabloda mevcut d
    egildir
617.                else {
618.                    //printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda ta
    mamlanmistir.\n",i);
619.                    return 1000; //adres olamayacak bir deger dondurulur
620.                }
621.            }
622.        }
623.
624.
625.        int main() {
626.
627.            //FILE *table_file, *table_file_2;  //hashtable'ların okunacağı fileler i
    cin file pointerlar
628.            char *file_name = (char*) calloc(TEXT_NAME_SIZE,sizeof(char));  //input d
    osyasinin adi
629.            struct node sozluk[M]; //sozluk icin hash tablosu
630.            struct node_2 hatali_kelimeler[M]; //hatali kelime sozlugu icin hash tabl
    osu
631.            int total_word_count_1 = 0; //tablodaki kelime sayisini tutan degisken
632.            int total_word_count_2 = 0;
633.            int kontrol;    //input file'in okunup okunamadiğinin kontrolunu tutan de
    gisken
634.            int i;
635.
636.            //tablolar 0'dan olusturuldugunda ilklendirme foru
637.            for(i = 0; i < M; i++) {
638.                hatali_kelimeler[i].count = 0;
639.                hatali_kelimeler[i].loadFactor = 0;
640.                sozluk[i].count = 0;
641.                sozluk[i].loadFactor = 0;
642.            }
```

```
643.
644.
645.        //  //sozulugun bulundugu hash tablosunun oldugu dosya acilir
646.        //    if ((table_file = fopen(SOZLUK_TXT, "rb")) == NULL) {
647.        //        printf("Error opening file\n");
648.        //        return 1;
649.        //    }
650.        //    //hash tablosu dosyadan okunur
651.        //    else {
652.        //      fread(sozluk, sizeof(struct node) * M, 1, table_file);  //tabloyu oku
653.        //        fclose(table_file);   //dosyayi kapat
654.        //  }
655.        //  total_word_count_1 =  sozluk[0].loadFactor * (float) M;
656.        //  printf("Sozluk basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablonun doluluk orani (load factor) : %.3f\n\n",M,sozluk[0].loadFactor);
657.        //  system("pause");
658.        //-------------------------------------------------------------------------------------------------------------------------
659.        //  //hatali kelime sozlugunun bulundugu hash tablosunun oldugu dosya acilir
660.        //  if ((table_file_2 = fopen(HATALI_TXT, "rb")) == NULL) {
661.        //        printf("Error opening file\n");
662.        //        return 1;
663.        //    }
664.        //    //hash tablosu dosyadan okunur
665.        //    else {
666.        //      fread(hatali_kelimeler, sizeof(struct node_2) * M, 1, table_file_2);
       //tabloyu oku
667.        //        fclose(table_file_2); //dosyayi kapat
668.        //  }
669.        //  total_word_count_2 =  hatali_kelimeler[0].loadFactor * (float) M;
670.        //  printf("\nHatali kelime sozlugu basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablonun doluluk orani (load factor) : %.3f\n\n",M,hatali_kelimeler[0].loadFactor);
671.        //  system("pause");
672.
673.
674.
675.            printf("Sozluk dosyasinin adini .txt uzantili olacak sekilde giriniz : ");
676.            scanf("%s",file_name);
677.            printf("\n");
678.            printf("Sozluk dosyadan okunuyor...\n");
679.            kontrol = readInputFile(file_name,&total_word_count_1,sozluk);  //okuma fonksiyonu cagirilir
680.            if(kontrol == 1) {
681.                printf("Input dosyasi okunamadi!\n");
682.                return 1; //dosya okunamazsa mainden cikilir
683.            }
684.            else {
685.                printf("Sozluk basari ile okundu.\nSozluk tablosunun doluluk orani = %.3f\n\n",sozluk[0].loadFactor);
686.            }
687.
688.
689.
690.        //  //hash tablosunun oldugu dosya acilir
691.        //    if ( (table_file = fopen(SOZLUK_TXT, "wb")) == NULL ) {
692.        //        printf("Error opening file\n");
693.        //        return 1;
694.        //    }
695.        //    //sozlugun hash tablosu dosyaya yazilir
696.        //  else {
697.        //          fwrite(sozluk, sizeof(struct node) * M, 1, table_file); //tablo dosyaya yazilir
```

```
698.        //          fclose(table_file); //dosya kapatilir
699.        //          printf("\nSozluk Dosyaya Basari ile Yazildi!\nTablonun doluluk or
    ani (load factor) : %.3f\n",sozluk[0].loadFactor);
700.        //  }
701.        //-------------------------------------------------------------------------
    ----------------------------------------------------------------
702.        //  if ( (table_file_2 = fopen(HATALI_TXT, "wb")) == NULL ) {
703.        //        printf("Error opening file\n");
704.        //        return 1;
705.        //    }
706.        //    //hatali kelime sozlugunun hash tablosu dosyaya yazilir
707.        //  else {
708.        //          fwrite(hatali_kelimeler, sizeof(struct node_2) * M, 1, table_file
    _2); //tablo dosyaya yazilir
709.        //          fclose(table_file_2);   //dosya kapatilir
710.        //          printf("\nHatali Kelimeler Dosyaya Basari ile Yazildi!\nTablonun
    doluluk orani (load factor) : %.3f\n",hatali_kelimeler[0].loadFactor);
711.        //  }
712.
713.            clear_input_buffer(); //scanf'den sonra input buffer temizlenir
714.            printf("\n");
715.            levenshtein(sozluk,hatali_kelimeler,&total_word_count_2);   //levenshtein
     fonksiyonu çağırılarak cümleler alınmaya ve işlenmeye başlanır
716.
717.            free(file_name); //free islemi
718.            return 0;   //end of main
719.        }
```