01.12.2020

YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM3021 ALGORİTMA ANALİZİ 2.ÖDEV RAPORU

Hashing Yöntemi ile Bir Kelimenin Geçtiği Dokümanları Listeleyen Sistem Tasarımı

AHMET SAİD SAĞLAM

17011501

KONU

Bir Kelimenin Geçtiği Dokümanları Listeleyen Sistem Tasarımı

Bu ödevde, yeni gelen bir dokümandaki kelimelerin hashing yöntemi ile bir sözlüğe yerleştirilmesi ve bir kelime arandığında yine hashing yöntemi ile sözlükte aranarak içinde yer aldığı dokümanların bulunması istenmiştir.

ÇÖZÜM

Kütüphane Eklenmesi ve Makro Atanması

```c
/*
@file
BLM3021 2020-2021 GUZ ODEV-2
Bu programda M elemanli bir hash tablosu dosyadan okunarak, input dosyalarindan hash tablosuna kelimeler eklenme modulu ve tabloda kelime arama modulu gerceklenmistir.
Programdan cikilirken guncel hash tablosu, okundugu dosyaya geri yazilir.

@author
İsim: Ahmet Said SAĞLAM
Öğrenci No: 17011501
Tarih: 01.12.2020
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 MM 996 //double probing hesabinda, iki numarali keyi hesaplarken kullanilacak olan sayi
#define M 997 //hash tablosunun boyutu
#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 31    //horner numarasi hesaplanirken kullanilcak olan asal sayi
#define WORD_SIZE 20    //alinan bir kelimenin maksimum uzunlugu
#define TEXT_NUMBER 20 //en fazla kac adet text eklenebilir tutan makro
#define HASH_TXT "17011501.txt" //hash tablosunun kaydedildigi dosyanin ismi
```

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

Yapı Tanımı ve getHorner Fonksiyonu

```c
//hash tablosunun her bir gozunde tutalacak olan yapi
typedef struct node {
    int text_count; //kelimenin kac adet dosyada gectigini tutan degisken
    float loadFactor;   //tablonun doluluk orani
    char word[WORD_SIZE]; //tutulan kelime
    char texts[TEXT_NUMBER][TEXT_NAME_SIZE]; //kelimenin gectigi textleri tutan matris

} node;

//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;

}
```

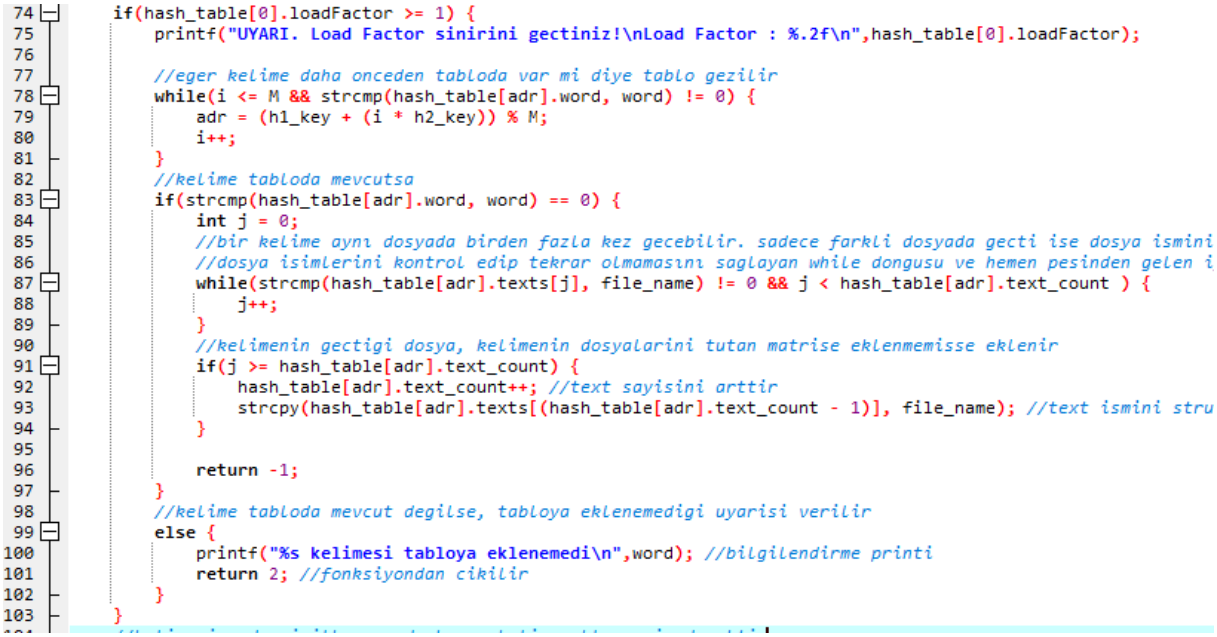
struct node yapısı hash tablosunun her bir elemanın içinde hangi bilgiler olduğunu gösteren ve tutan yapıdır. Tablonun her bir gözünde, o gözde bulunabilecek kelimenin kendisini tutan karakter dizisi, kelimenin kaç adet dosyada geçtiğini tutan count değeri, kelimenin geçtiği dosyaların isimlerini tutan bir matris ve son olarak da tablonun doluluk oranını tutan loadFactor değeri bulunur.

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

insertTable Fonksiyonu ve Hash Tablosuna Kelime Ekleme İşlemleri

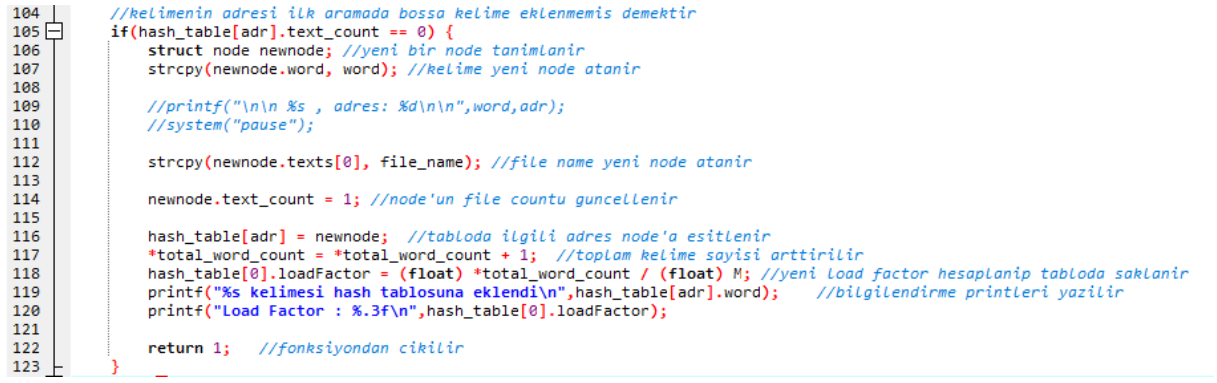
```c
//dosyadan okunan satirdaki kelimeyi alip tabloya ekleyen fonksiyon
int insertTable(char *word, char *file_name, int *total_word_count, struct node *hash_table) {
    int i = 0; //adres hesabinda adim sayisi
    long long key = getHorner(word);    //kelimenin horner sayisi
    int adr;    //kelimenin tablodaki adresi

    //adres double probing yontemi ile hesaplanir
    int h1_key = key % M;
    int h2_key = 1 + (key % MM);
    adr = (h1_key + (i * h2_key)) % M;
    i++;    //adim sayisi olasi bir yeni hesaplama icin guncellenir

    //printf("\n\n %s , adres: %d\n\n",word,adr);
    //system("pause");

    //load factor 0.8'in ustundeyse uyari verilir
    if(hash_table[0].loadFactor > 0.8 && hash_table[0].loadFactor < 1) {
        printf("UYARI. Load Factor sinirina yaklasiyorsunuz!\nLoad Factor : %.2f\n",hash_table[0].loadFactor);
    }
```

Fonksiyonun içerisine tabloya eklenmesi amaçlanan kelimeyi, kelimenin geçtiği text'in adını, tablodaki toplam kelime sayısını ve son olarak da tablonun kendisini alır. İlk etapta kelimenin sayısal karşılığı getHorner fonksiyonu çağırılarak hesaplanır. Sonrasında bu horner sayısı kullanılarak ödev dokümanında da bahsedildiği üzere double hashing yöntemiyle kelimenin adresi hesaplanır. Sonrasında tablonun doluluk oranı kontrol edilir ve 0.8'in üzerindeyse kullanıcıya uyarı verilir.



```
74      if(hash_table[0].loadFactor >= 1) {
75          printf("UYARI. Load Factor sinirini gectiniz!\nLoad Factor : %.2f\n",hash_table[0].loadFactor);
76
77          //eger kelime daha onceden tabloda var mi diye tablo gezilir
78          while(i <= M && strcmp(hash_table[adr].word, word) != 0) {
79              adr = (h1_key + (i * h2_key)) % M;
80              i++;
81          }
82          //kelime tabloda mevcutsa
83          if(strcmp(hash_table[adr].word, word) == 0) {
84              int j = 0;
85              //bir kelime aynı dosyada birden fazla kez gecebilir. sadece farkli dosyada gecti ise dosya ismini
86              //dosya isimlerini kontrol edip tekrar olmamasını saglayan while dongusu ve hemen pesinden gelen i
87              while(strcmp(hash_table[adr].texts[j], file_name) != 0 && j < hash_table[adr].text_count ) {
88                  j++;
89              }
90              //kelimenin gectigi dosya, kelimenin dosyalarini tutan matrise eklenmemisse eklenir
91              if(j >= hash_table[adr].text_count) {
92                  hash_table[adr].text_count++; //text sayisini arttir
93                  strcpy(hash_table[adr].texts[(hash_table[adr].text_count - 1)], file_name); //text ismini stru
94              }
95
96              return -1;
97          }
98          //kelime tabloda mevcut degilse, tabloya eklenemedigi uyarisi verilir
99          else {
100             printf("%s kelimesi tabloya eklenemedi\n",word); //bilgilendirme printi
101             return 2; //fonksiyondan cikilir
102         }
103     }
```

Fonksiyonun sonraki aşamasında ise tablonun doluluk oranının 1’ ulaşıp ulaşmadığı yani tablonun dolup dolmadığı kontrol edilir. Tablo eğer dolduysa fonksiyon içerisine aldığı kelime halihazırda tabloda var mı diye kontrol eder. Eğer varsa tabloda kelimenin bulunduğu göze text ismini ekler. Eğer kelime tabloda yoksa kelimeyi ekleyemediğini kullanıcıya bildirir. Son adım olarak da fonksiyondan uygun dönüş değeri ile çıkılır.



```
104     //kelimenin adresi ilk aramada bossa kelime eklenmemis demektir
105     if(hash_table[adr].text_count == 0) {
106         struct node newnode; //yeni bir node tanimlanir
107         strcpy(newnode.word, word); //kelime yeni node atanir
108
109         //printf("\n\n %s , adres: %d\n\n",word,adr);
110         //system("pause");
111
112         strcpy(newnode.texts[0], file_name); //file name yeni node atanir
113
114         newnode.text_count = 1; //node'un file countu guncellenir
115
116         hash_table[adr] = newnode;  //tabloda ilgili adres node'a esitlenir
117         *total_word_count = *total_word_count + 1;  //toplam kelime sayisi arttirilir
118         hash_table[0].loadFactor = (float) *total_word_count / (float) M; //yeni load factor hesaplanip tabloda saklanir
119         printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].word);    //bilgilendirme printleri yazilir
120         printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
121
122         return 1;   //fonksiyondan cikilir
123     }
```

Tablo henüz dolmadıysa, kelime kendisi için belirlenmiş adreste aranır. Adres boşsa kelime ve text bilgisi tabloya eklenir. Load factor güncellenir ve fonksiyondan çıkılır.

```
124     else {
125         //ilk aramada hash tablosunun gözü doluysa kelime kontrolü yapılarak ve gerekirse yeni adres hesaplanarak ilerlenir
126         while(hash_table[adr].text_count != 0 && strcmp(hash_table[adr].word, word) != 0) {
127             adr = (h1_key + (i * h2_key)) % M;
128             i++;
129         }
130         //whiledan kelime ile karsilasildigi icin cikildiysa, dosya ismi eklenmek uzere if kosuluna girilir
131         if(strcmp(hash_table[adr].word, word) == 0) {
132             int j = 0;
133             //bir kelime aynı dosyada birden fazla kez gecebilir. sadece farkli dosyada gecti ise dosya ismini tabloda ilgili yere eklemelidir
134             //dosya isimlerini kontrol edip tekrar olmamasını saglayan while dongusu ve hemen pesinden gelen if kosulu
135             while(strcmp(hash_table[adr].texts[j], file_name) != 0 && j < hash_table[adr].text_count ) {
136                 j++;
137             }
138             //kelime bu dosyada ilk kez geciyorsa tabloya dosya ismi eklenir
139             if(j >= hash_table[adr].text_count) {
140                 hash_table[adr].text_count++; //text sayisini arttir
141                 strcpy(hash_table[adr].texts[(hash_table[adr].text_count - 1)], file_name); //text ismini structa kopyala
142             }
143 
144             return -1; //fonksiyondan cikilir
145         }
```

Tablo henüz dolmadıysa ancak kelimenin kendisi için belirlenmiş adreste bir kelime varsa, öncelikle kelimenin kendisi mi değil mi diye kontrol edilir. Kelimenin kendisiyse kelime daha önceden tabloya eklenmiş demektir ve sadece kelimenin geçtiği text ismi, tekrar tekrar eklenmemesi adına kontrol edilerek tabloda kelimenin adresinde bulunan “texts” matrisine eklenir ve fonksiyondan çıkılır. Kelimenin kendisi değil de başka bir kelime ise kelime ile karşılaşılana kadar veya da boş bir adres ile karşılaşılana kadar kelimenin adres değeri double hashing yöntemine göre güncellenir. Oluşturulan yeni adres değerlerine göre yeniden aramalar yapılır. Eğer arama sonucunda tabloda kelimenin kendisine rastlanırsa, yine kelime daha önceden tabloya eklenmiş demektir ve sadece kelimenin geçtiği text ismi, tekrar tekrar eklenmemesi adına kontrol edilerek tabloda kelimenin adresinde bulunan “texts” matrisine eklenir ve fonksiyondan çıkılır.

```
146         //whiledan bos adrese gelindigi icin cikildiysa
147         else {
148             //yeni bir node tanimlanir ve veriler node kopyalanir
149             struct node newnode;
150             strcpy(newnode.word, word);
151             strcpy(newnode.texts[0], file_name);
152             newnode.text_count = 1;
153 
154             hash_table[adr] = newnode; //olusturulan node tabloda ilgili adrese atilir
155             *total_word_count = *total_word_count + 1; //tablodaki kelime sayisi guncellenir
156             hash_table[0].loadFactor =(float) *total_word_count / (float) M; //load factor hesaplanir
157             printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].word); //bilgilendirme printleri atilir
158             printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
159             //printf("count : %d\n",*total_word_count);
160 
161             return 1; //fonksiyondan cikilir
162         }
163     }
164 }
```

Oluşturulan yeni adres değerlerine göre yapılan aramalar sonucu tabloda boş bir adrese denk gelinirse kelimenin ilk oluşturulan adres veya adreslerinde başka kelimeler mevcut demektir. Yeni karşılaşılan boş adrese kelime ve text bilgisi eklenir. Load factor güncellenir ve fonksiyondan çıkılır.

readInputFile Fonksiyonu ve Dosyadan Veri Okuma İşlemleri

```
168 //dosyadan veriyi satir satir okuyan fonksiyon
169 int readInputFile(char *file_name, int *total_word_count, struct node *hash_table) {
170     FILE *inputFile; //input file 'ı açmaya yarayan file pointer
171     char ch; //dosyadan karakterler okunup bu degiskene aktarilir(satir sayisini hesaplamak icin)
172     char *buffer = (char*) calloc(BUFFER_SIZE,sizeof(char)); //dosyadan alinan satirin tutuldugu buffer
173     char *word  = (char*) calloc (WORD_SIZE,sizeof(char));//satirdan parcalanip alinan kelime
174     char *org_word = (char*) calloc (WORD_SIZE,sizeof(char));//orjinal kelimeyi tutar
175     int line_count = 0; //dosyadaki satir sayisini tutan degisken
176     int i, j; //dongu degiskeni
177     int isExist; //kelime hash table'da var mi yok mu donus degerini tutan degisken
178     //int wordCount = total_word_count;
179 
180     if((inputFile = fopen(file_name,"r")) == NULL) {
181         printf("Dosya okunmak icin acilamadi!\n");
182         return 1;
183     }
184     else {
185         //dosyadaki satir sayisini hesaplayan do-while bloğu
186         do
187         {
188             ch = fgetc(inputFile); //karakter oku
189             //new line ise line_count'u 1 arttır
190             if (ch == '\n') {
191                 line_count++;
192             }
193         } while (ch != EOF); //dosya sonuna gelene kadar
194         rewind(inputFile); //dosyada basa don
195         line_count++; //line count son haline guncellenir
```

readInputFile fonksiyonu içerisine açıp okuyacağı input dosyasının ismini, hash tablosundaki toplam kelime sayısını ve hash tablosunun kendisini alır. İlk adımda input dosyasını açmaya çalışır ve bir hatayla karşılaşırsa uyarı verip fonksiyondan çıkar. Bir hatayla karşılaşılmadığı durumda dosya açılır ve dosyadaki satır sayısı hesaplanır.

```
197         //dosyadan veriler satir satir okunur ve isleme alinir
198         for(i = 0; i < line_count; i++) {
199             fgets(buffer,BUFFER_SIZE * sizeof(char),inputFile); //satırı dosyadan buffer'a al
200             buffer[strlen(buffer)] = '\0'; //bufferin sinirini belirle
201             printf("-------------------------------------------------\n");
202             printf("Okunan satir : %s\n\n",buffer);
203             system("PAUSE");
204             printf("\n");
205             word = strtok(buffer, AYRAC); //satirdaki ilk kelime alinir
206             word[strlen(word)] = '\0'; //kelime sinirini belirle
207             strcpy(org_word,word); //alinan orjinal kelime org_word'de saklanir
208             //case insensitive durum saglanmasi icin kelimedeki butun harfler kucuk harfe cevrilir
209             for(j = 0; j <strlen(word); j++) {
210                 word[j] = tolower(word[j]);
211             }
212 //          printf("word : %s\n",word);
213 //          system("PAUSE");
214             isExist = insertTable(word,file_name,total_word_count,hash_table); //kelime hash tablosuna eklenmek uzere insertTable fonksiyonu cagirilir
215             //donus degeri kontrol edilir
216             if(isExist == -1) {
217                 printf("%s kelimesi zaten tabloda mevcut!\n\n",org_word); //kelime hash tablosunda mevcutsa bilgilendirme printi atilir
218             }
```

Sonraki adımda dosyadaki veriler satır satır okunarak bir buffer'a alınır. Her satır okuma işleminde buffer'daki ilk kelime strtok fonksiyonu yardımıyla boşluğa göre parçalanır ve alınır. Orijinal kelime bir dizide tutulur ve alınan kelimenin harfleri tolower fonksiyonu yardımıyla, dokümanda da bahsedildiği üzere case insensitive durumu sağlamak amacıyla küçük harflere dönüştürülür. Ardından insertTable fonksiyonu çağırılır ve kelime hash tablosuna eklenmeye çalışılır. Fonksiyonun dönüş değerine göre kelime tabloda mevcutsa uyarı printi verilir.

```
            word = strtok(NULL, AYRAC); //satirdaki sonraki kelime alinir
            //satirdaki diger kelimeler parcalara ayrilarak ayristirilir
            while( word != NULL ) {
                //satir sonuna gelindiyse new line karakter maskelenir
                if(word[strlen(word)-1] == '\n') {
                    word[strlen(word)-1] = '\0';
                }
                strcpy(org_word,word); //alinan orjinal kelime org_word'de saklanir
                //case insensitive durum saglanmasi icin kelimedeki butun harfler kucuk harfe cevrilir
                for(j = 0; j <strlen(word); j++) {
                    word[j] = tolower(word[j]);
                }
                word[strlen(word)] = '\0';  //kelime siniri belirlenir
//              printf("word : %s, size %d\n",word,strlen(word));
//              system("PAUSE");
                isExist = insertTable(word,file_name,total_word_count,hash_table); //kelime hash tablosuna eklenmek uzere insertTable fonksiyonu cagirilir
                if(isExist == -1) {
                    printf("%s kelimesi zaten tabloda mevcut!\n\n",org_word); //kelime hash tablosunda mevcutsa bilgilendirme printi atilir
                }
//              else if(isExist == 1) {
//                  //printf("%s kelimesi eklendi!\n\n",org_word);
//              }
                word = strtok(NULL, AYRAC); //satirdaki diger kelimeler alinir
            }

        }
        fclose(inputFile); //dosya kapatilir
    }
    //free islemleri
    free(buffer);
    free(word);
    free(org_word);
    return 0; //fonksiyondan cikilir
}
```

Sonrasında bu işlem buffer'daki diğer tüm kelimeler için tekrarlanır. Dosyadaki tüm satırların okunmasının ardından ise dosya kapatılır, free işlemleri yapılır ve fonksiyondan çıkılır.

searchHash Fonksiyonu ve Tabloda Kelime Aranması İşlemleri

```
//icine verilen kelimeyi hash tablosunda arayan fonksiyon
void searchHash (struct node *hash_table) {
    char *word = (char*) calloc(WORD_SIZE, sizeof(char)); //kullanicadan alinan ve tabloada aranacak olan kelime
    int cont = 1; //yeni arama olup olmayacagini kontrol eden degisken
    int i = 0; //kelimenin tablodaki adresi double probinge gore hesaplanirken adim sayisini tutan degisken
    long long key; //kelimenin horner methoduna gore key degeri
    int adr; //kelimenin tablodaki adresi
    int j;  //dongu degiskeni
    while(cont) {
        printf("\nAramak istediginiz kelimeyi giriniz : \n");
        scanf("%s",word);   //aranmak istenen kelime kullanicidan alinir
        word[strlen(word)] = '\0';  //kelime siniri belirlenir
        //kelime case insensitive'lik saglanmasi icin kucuk harflere cevirilir
        for(j = 0; j <strlen(word); j++) {
            word[j] = tolower(word[j]);
        }
//      printf("kontrol!!! kelime : %s\n",word);
//      system("pause");
        printf("\n");
        key = (long long) getHorner(word);  //horner sayisi alinir

        //kelimenin tablodaki adresi double probing yontemiyle hesaplanir
        int h1_key = key % M;
        int h2_key = 1 + (key % MM);
        adr = (h1_key + (i * h2_key)) % M;
        i++;  //adim sayisi guncellenir
```

searchHash fonksiyonu içerisine üzerinde arama yapılacak hash tablosunu alır. Aranacak kelime için yer açılır ve kullanıcının sürekli arama yapabilmesini sağlamak amacıyla while döngüsüne girilir. Sonrasında kullanıcıdan aramak istediği kelime alınır ve kelimenin horner sayısı hesaplanarak olası adresi de hesaplanır.

```
284         //kelime tabloda mevcut degilse ilgili gozde bulunan struct'ın text_count degeri 0 demektir.
285         if(hash_table[adr].text_count == 0) {
286             printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda tamamlanmistir.\n\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz.\n",i);
287             scanf("%d",&cont);
288         }
289         //kelimenin tabloda mevcut olma ihtimalinde
290         else {
291             //kelime ile karsilasana kadar veya bos goz gorene kadar donen while
292             while(hash_table[adr].text_count != 0 && strcmp(hash_table[adr].word, word) != 0) {
293                 adr = (h1_key + (i * h2_key)) % M;  //adres degeri her adımda guncellenir
294                 i++;    //adim sayisi guncellenir
295             }
296             //kelime ile karsilasildigi  icin while'dan cikilmissa kelime tabloda mevcuttur
297             if(strcmp(hash_table[adr].word, word) == 0) {
298                 printf("Kelime tabloda mevcut!\nKelime toplamda %d adet dosyada gecmistir.\nKelimenin gectigi dosyalar : \n",hash_table[adr].text_count);
299                 //kelimenin gectigi dosyalari ekrana yazdiran for
300                 for(j = 0; j < hash_table[adr].text_count; j++) {
301                     printf("%d. dosyanin ismi : %s\n",(j+1),hash_table[adr].texts[j]);
302                 }
303                 printf("Arama islemi %d adimda tamamlanmistir.\n",i);
304                 printf("\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz.\n");
305                 scanf("%d",&cont);
306             }
307             //bos goze gelindigi icin whiledan cikilmissa kelime tabloda mevcut degildir
308             else {
309                 printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda tamamlanmistir.\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz.\n",i);
310                 scanf("%d",&cont);  //yeni arama yapilacak mi bilgisi kullanicidan alinir
311             }
312
313         }
```

Hesaplanan adrese göre yapılan ilk aramada eğer adres boş ise kelimede tabloda mevcut değildir ve bununla ilgili bilgi ekrana yazdırılır. Adres doluysa bu adreste aranan kelime mi yoksa başka farklı bir kelime mi olduğu kontrolü yapılır. Aranan kelimeye rastlanılırsa kelimenin bulunduğu bilgisi, kelimeyi barındıran text sayısı ve isimleri ile beraber kelimeye kaç arama adımında ulaşıldığının bilgisi ekrana yazdırılır. Kelimeye rastlayamadan boş bir adrese gelindi ise kelime tabloda mevcut değil demektir ve bununla ilgili bilgi ekrana yazdırılır.

```
314             //yeni aramalar icin word'u temizleyen for dongusu
315             for(j = 0; j < WORD_SIZE; j++) {
316                 word[j] = '\0';
317             }
318             i = 0; //yeni aramada adim sayisi 0'dan baslatilir
319         }
320         free(word);  //free islemi
321     }
```

Sonrasında kullanıcıdan aramaya devam edip etmeyeceği bilgisi alınır ve alınan cevaba göre arama işlemleri devam eder veya free işleminin ardından fonksiyon işlevini sonlandırır.

Ana Fonksiyon

```c
int main() {

    FILE *table_file;   //hashtable'ın okunacağı file icin file pointer
    char *file_name = (char*) calloc(TEXT_NAME_SIZE,sizeof(char));  //input dosyasinin adi
    struct node hashTable[M]; //hash table
    int total_word_count = 0;   //tablodaki kelime sayisini tutan degisken
    int control;    //secim kontrolu - dosyadan kelime okuma veya tablodan kelime arama
    int kontrol;    //input file'in okunup okunamadiğinin kontrolunu tutan degisken
    //int i;

//  //tablo 0'dan olusturuldugunda ilklendirme foru
//  for(i = 0; i < M; i++) {
//      hashTable[i].text_count = 0;
//      hashTable[i].loadFactor = 0;
//  }

    //hash tablosunun oldugu dosya acilir
    if ((table_file = fopen(HASH_TXT, "rb")) == NULL) {
        printf("Error opening file\n");
        return 1;
    }
    //hash tablosu dosyadan okunur
    else {
        fread(hashTable, sizeof(struct node) * M, 1, table_file);   //tabloyu oku
        fclose(table_file); //dosyayi kapat
    }
    total_word_count =  hashTable[0].loadFactor *(float) M;
    printf("Hash Table basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablonun doluluk orani (load factor) : %.3f\n\n",M,hashTable[0].loadFactor);
```

main fonksiyonda gerekli tanımlamaların ardından hash tablosu dosyadan okunmaya çalışılır. Eğer okunamazsa hata yazısıyla beraber program kendini kapatır. Başarılı bir okuma gerçekleşirse kullanıcıya bu bildirilir ve tablo boyutu ile beraber tablonun doluluk oranı ekrana yazdırılır.

```c
    //kullanicidan arama veya ekleme islemi icin istegi alinir
    printf("Hash Tablosunda arama yapmak icin 1'e, tabloya kelime eklemek icin 2'ye basiniz.\n");
    scanf("%d",&control);

    if(control == 1) {
        searchHash(hashTable);  //arama fonksiyonu cagirlir

    }
    else if(control == 2) {
        printf("Input dosyasinin adini .txt uzantili olacak sekilde giriniz : ");
        scanf("%s",file_name);
        printf("\n");
        kontrol = readInputFile(file_name,&total_word_count,hashTable); //okuma fonksiyonu cagirilir
        if(kontrol == 1) {
            printf("Input dosyasi okunamadi!\n");
            return 1; //dosya okunamazsa mainden cikilir
        }
//      printf("\n\n\nMAINDEYIZ -- total word: %d,load factor : %f\n\n\n",total_word_count,hashTable[0].loadFactor);
//      system("pause");
    }
```

Sonraki adımda kullanıcıdan tabloya veri eklemek mi veya tabloda kelime aramak mı istediği bilgisi alınır ve ona göre gerekli fonksiyonlar çağırılır. Eğer kullanıcı tabloya veri eklemek isterse input dosyasının adını burada programa vermelidir. Input dosyasının açılışında bir sorun olursa program hata printi verir ve sonlanır.

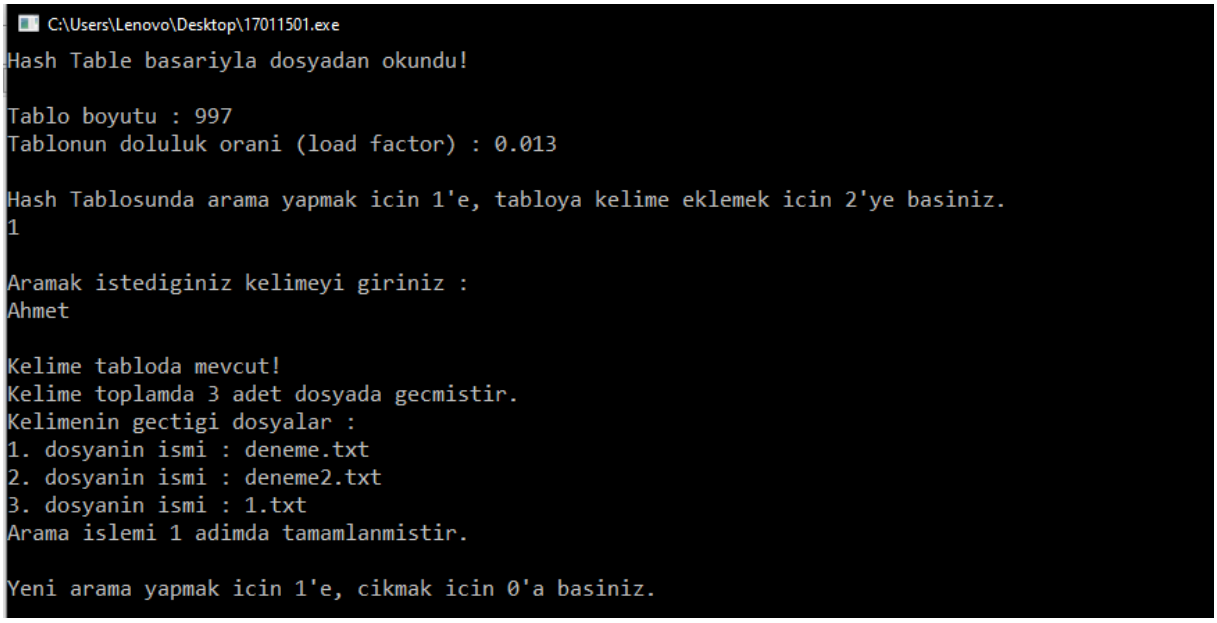
```
375     //hash tablosunun oldugu dosya acilir
376     if ( (table_file = fopen(HASH_TXT, "wb")) == NULL ) {
377         printf("Error opening file\n");
378         return 1;
379     }
380     //hash tablosu dosyaya yazilir
381     else {
382             fwrite(hashTable, sizeof(struct node) * M, 1, table_file); //tablo dosyaya yazilir
383             fclose(table_file); //dosya kapatilir
384             printf("\nTablo Dosyaya Basari ile Yazildi!\nTablonun doluluk orani (load factor) : %.3f\n",hashTable[0].loadFactor);
385     }
386
387
388     free(file_name); //free islemi
389     return 0;   //end of main
390 }
```

Son olarak hash tablosunun güncel hali tablonun okunduğu dosyaya yazılır ve free işleminin ardından main fonksiyonuyla beraber program sonlanır.

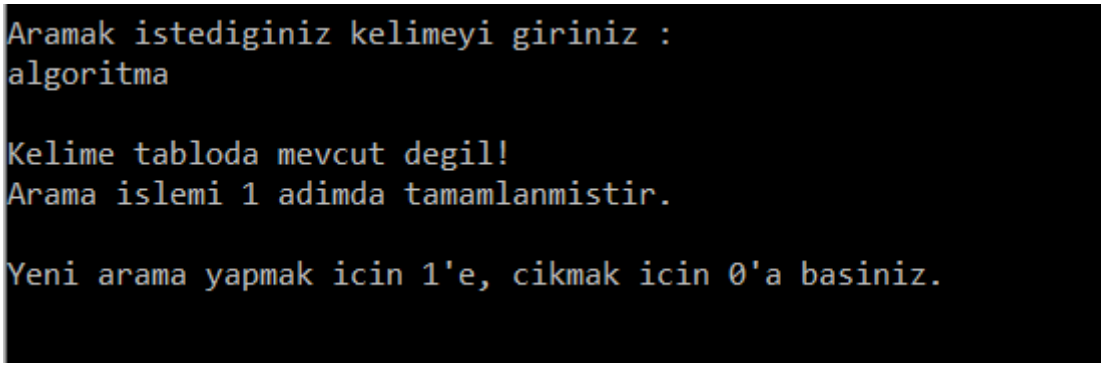
KARMAŞIKLIK ANALİZİ

Hashing yöntemi kullanılarak yapılan tabloya ekleme ve arama işlemlerinde average case karmaşıklık O(1)'dir. Bunun nedeni hesaplanan adrese ortalama durumda collusion olmadan erişip tek adımda arama ve ekleme yapma işlemlerinin tamamlanabilir olmasıdır. En kötü durumda ise karmaşıklık O(n) olacaktır. Sürekli collusionlar ile karşılaşıldığı durumda tablodaki tüm adreslerin gezilme ihtimalinden bu karmaşıklık ortaya çıkmaktadır.

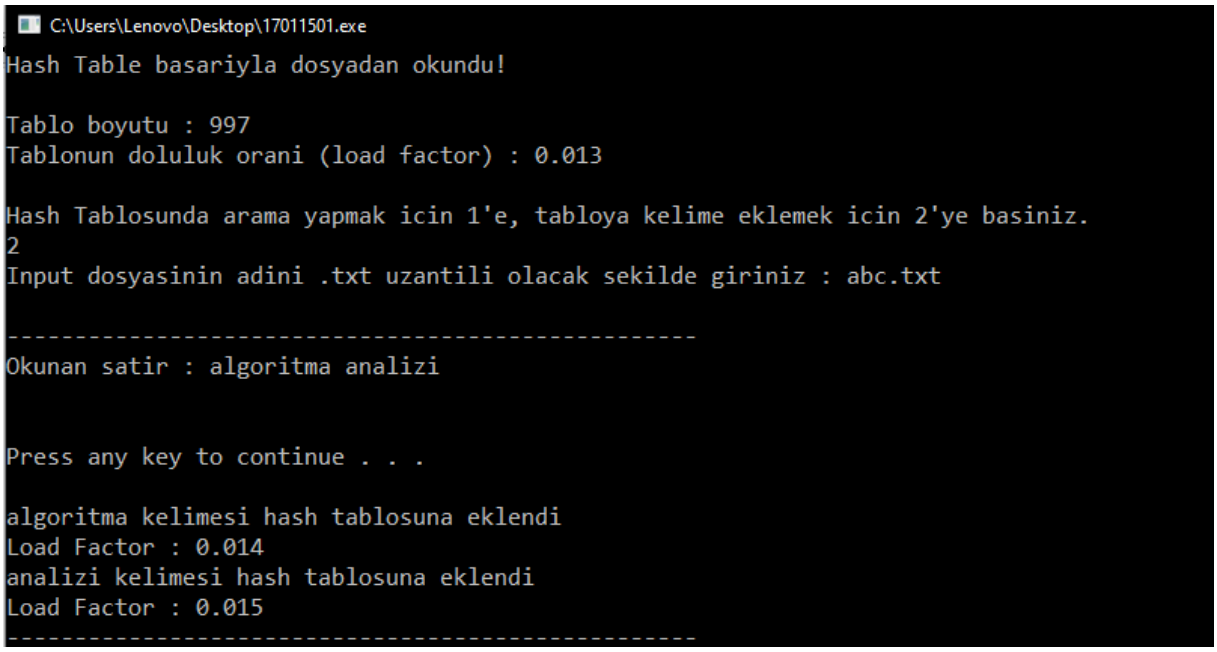
PROGRAM ÇIKTILARI



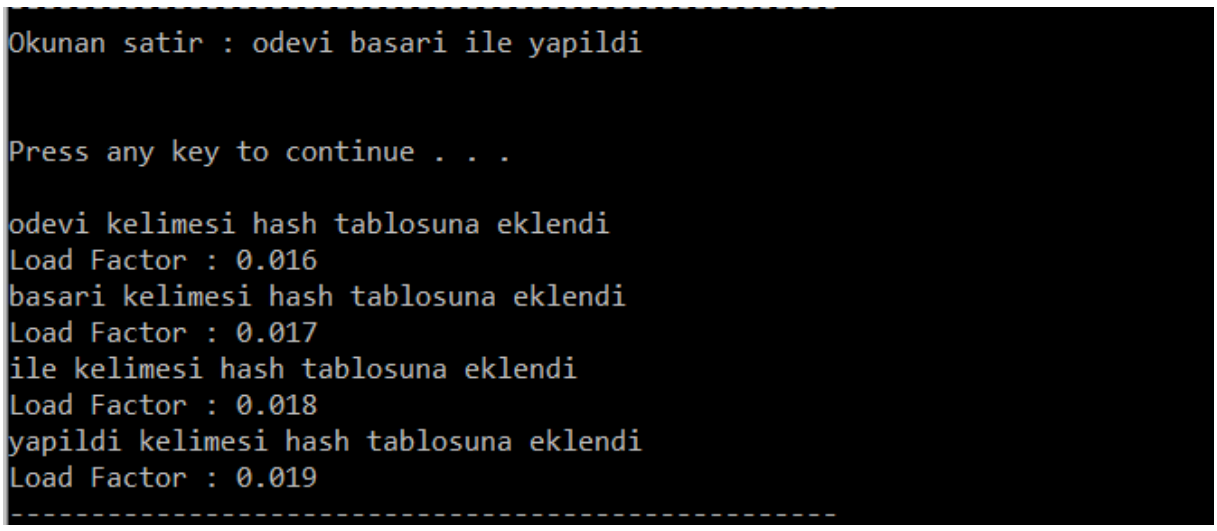
Tabloda Var Olan Bir Kelimenin Aranması



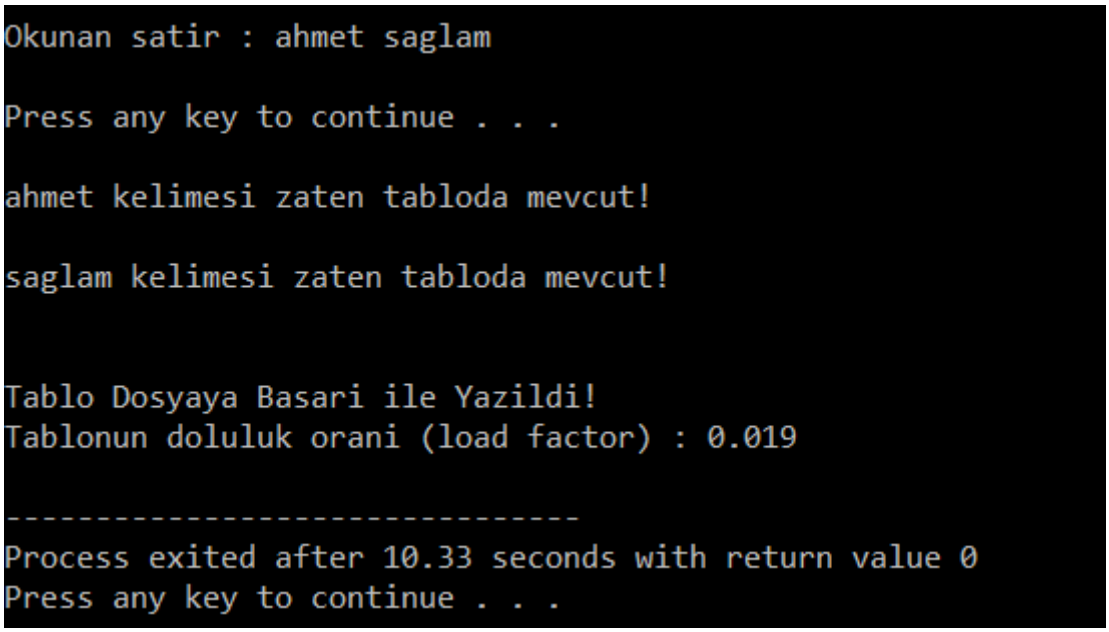
Tabloda Var Olmayan Bir Kelimenin Aranması



Tabloya Dosyadan Kelime Eklenmesi Durumu



Tabloya Dosyadan Kelime Eklenmesi Durumu



Tabloya Dosyadan Kelime Eklenmesi Durumu

SOURCE CODE

```
1.  /*
2.  @file
3.  BLM3021 2020-2021 GUZ ODEV-2
4.  Bu programda M elemanli bir hash tablosu dosyadan okunarak, input dosyalarindan hash
     tablosuna kelimeler eklenme modulu ve tabloda kelime arama modulu gerceklenmistir.
5.  Programdan cikilirken guncel hash tablosu, okundugu dosyaya geri yazilir.
6.
7.  @author
8.  İsim: Ahmet Said SAĞLAM
9.  Öğrenci No: 17011501
10. Tarih: 01.12.2020
11. E-Mail: l1117501@std.yildiz.edu.tr
12. Compiler: TDM-GCC 4.9.2 64 bit-Release
13. IDE: DEV-C++ (version 5.11)
14. İşletim Sistemi: Windows 10 Pro 64 bit
15. */
16.
17. #include <stdio.h>
18. #include <stdlib.h>
19. #include <string.h>
20. #include <conio.h>
21. #include <stdbool.h>
22. #include <ctype.h>
23. #define MM 996 //double hashing hesabinda, iki numarali keyi hesaplarken kullanilaca
    k olan sayi
24. #define M 997 //hash tablosunun boyutu
25. #define BUFFER_SIZE 10000 //txt dosyadan alınan satırın saklanacağı bufferin boyutu

26. #define TEXT_NAME_SIZE 20 // max file ismi uzunlugu
27. #define AYRAC " "   //kelimelerin ayrilacagi delim ifadesi
28. #define HORNER_NUMBER 31    //horner numarasi hesaplanirken kullanilcak olan asal sa
    yi
29. #define WORD_SIZE 20    //alinan bir kelimenin maksimum uzunlugu
30. #define TEXT_NUMBER 20 //en fazla kac adet text eklenebilir tutan makro
31. #define HASH_TXT "17011501.txt" //hash tablosunun kaydedildigi dosyanin ismi
32.
33. //hash tablosunun her bir gozunde tutalacak olan yapi
34. typedef struct node {
35.     int text_count; //kelimenin kac adet dosyada gectigini tutan degisken
36.     float loadFactor;   //tablonun doluluk orani
37.     char word[WORD_SIZE]; //tutulan kelime
38.     char texts[TEXT_NUMBER][TEXT_NAME_SIZE]; //kelimenin gectigi textleri tutan matr
    is
39.
40. } node;
41.
42. //icine verilen kelimenin horner sayisini donduren fonksiyon
43. long long getHorner(char *word) {
44.     long long key = 0L; //kelimenin sayisal key karşılığını tutacak olan degisken
45.     int i; //dongu degiskeni
46.     //kelimenin horner sayisi hesaplanir
47.     for(i = 0; i < strlen(word); i++) {
48.         key = (long long) (HORNER_NUMBER * key + (word[i] - 'a' + 1));
49.     }
50.     return (long long) key;
51.
52. }
53.
54. //dosyadan okunan satirdaki kelimeyi alip tabloya ekleyen fonksiyon
55. int insertTable(char *word, char *file_name, int *total_word_count, struct node *has
    h_table) {
56.     int i = 0; //adres hesabinda adim sayisi
57.     long long key = getHorner(word);    //kelimenin horner sayisi
```

```
58.     int adr;    //kelimenin tablodaki adresi
59.
60.     //adres double probing yontemi ile hesaplanir
61.     int h1_key = key % M;
62.     int h2_key = 1 + (key % MM);
63.     adr = (h1_key + (i * h2_key)) % M;
64.     i++;    //adim sayisi olasi bir yeni hesaplama icin guncellenir
65.
66.     //printf("\n\n %s , adres: %d\n\n",word,adr);
67.     //system("pause");
68.
69.     //load factor 0.8'in ustundeyse uyari verilir
70.     if(hash_table[0].loadFactor > 0.8 && hash_table[0].loadFactor < 1) {
71.         printf("UYARI. Load Factor sinirina yaklasiyorsunuz!\nLoad Factor : %.2f\n",
    hash_table[0].loadFactor);
72.     }
73.     //tablo dolduysa uyari verilir
74.     if(hash_table[0].loadFactor >= 1) {
75.         printf("UYARI. Load Factor sinirini gectiniz!\nLoad Factor : %.2f\n",hash_ta
    ble[0].loadFactor);
76.
77.         //eger kelime daha onceden tabloda var mi diye tablo gezilir
78.         while(i <= M && strcmp(hash_table[adr].word, word) != 0) {
79.             adr = (h1_key + (i * h2_key)) % M;
80.             i++;
81.         }
82.         //kelime tabloda mevcutsa
83.         if(strcmp(hash_table[adr].word, word) == 0) {
84.             int j = 0;
85.             //bir kelime aynı dosyada birden fazla kez gecebilir. sadece farkli dosy
    ada gecti ise dosya ismini tabloda ilgili yere eklemelidir
86.             //dosya isimlerini kontrol edip tekrar olmamasını saglayan while dongusu
     ve hemen pesinden gelen if kosulu
87.             while(strcmp(hash_table[adr].texts[j], file_name) != 0 && j < hash_table
    [adr].text_count ) {
88.                 j++;
89.             }
90.             //kelimenin gectigi dosya, kelimenin dosyalarini tutan matrise eklenmemi
    sse eklenir
91.             if(j >= hash_table[adr].text_count) {
92.                 hash_table[adr].text_count++; //text sayisini arttir
93.                 strcpy(hash_table[adr].texts[(hash_table[adr].text_count -
     1)], file_name); //text ismini structa kopyala
94.             }
95.
96.             return -1;
97.         }
98.         //kelime tabloda mevcut degilse, tabloya eklenemedigi uyarisi verilir
99.         else {
100.                printf("%s kelimesi tabloya eklenemedi\n",word); //bilgilendirme
    printi
101.                return 2; //fonksiyondan cikilir
102.            }
103.        }
104.        //kelimenin adresi ilk aramada bossa kelime eklenmemis demektir
105.        if(hash_table[adr].text_count == 0) {
106.            struct node newnode; //yeni bir node tanimlanir
107.            strcpy(newnode.word, word); //kelime yeni node atanir
108.
109.            //printf("\n\n %s , adres: %d\n\n",word,adr);
110.            //system("pause");
111.
112.            strcpy(newnode.texts[0], file_name); //file name yeni node atanir
113.
114.            newnode.text_count = 1; //node'un file countu guncellenir
115.
```

```
116.                hash_table[adr] = newnode;  //tabloda ilgili adres node'a esitlenir
117.                *total_word_count = *total_word_count + 1;  //toplam kelime sayisi ar
    ttirilir
118.                hash_table[0].loadFactor = (float) *total_word_count / (float) M; //y
    eni load factor hesaplanip tabloda saklanir
119.                printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].word);
       //bilgilendirme printleri yazilir
120.                printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
121.
122.                return 1;   //fonksiyondan cikilir
123.            }
124.            else {
125.                //ilk aramada hash tablosunun gözü doluysa kelime kontrolü yapılarak
    ve gerekirse yeni adres hesaplanarak ilerlenir
126.                while(hash_table[adr].text_count != 0 && strcmp(hash_table[adr].word,
     word) != 0) {
127.                    adr = (h1_key + (i * h2_key)) % M;
128.                    i++;
129.                }
130.                //whiledan kelime ile karsilasildigi icin cikildiysa, dosya ismi ekle
    nmek uzere if kosuluna girilir
131.                if(strcmp(hash_table[adr].word, word) == 0) {
132.                    int j = 0;
133.                    //bir kelime aynı dosyada birden fazla kez gecebilir. sadece fark
    li dosyada gecti ise dosya ismini tabloda ilgili yere eklemelidir
134.                    //dosya isimlerini kontrol edip tekrar olmamasını saglayan while
    dongusu ve hemen pesinden gelen if kosulu
135.                    while(strcmp(hash_table[adr].texts[j], file_name) != 0 && j < has
    h_table[adr].text_count ) {
136.                        j++;
137.                    }
138.                    //kelime bu dosyada ilk kez geciyorsa tabloya dosya ismi eklenir

139.                    if(j >= hash_table[adr].text_count) {
140.                        hash_table[adr].text_count++; //text sayisini arttir
141.                        strcpy(hash_table[adr].texts[(hash_table[adr].text_count -
     1)], file_name); //text ismini structa kopyala
142.                    }
143.
144.                    return -1; //fonksiyondan cikilir
145.                }
146.                //whiledan bos adrese gelindigi icin cikildiysa
147.                else {
148.                    //yeni bir node tanimlanir ve veriler node kopyalanir
149.                    struct node newnode;
150.                    strcpy(newnode.word, word);
151.                    strcpy(newnode.texts[0], file_name);
152.                    newnode.text_count = 1;
153.
154.                    hash_table[adr] = newnode; //olusturulan node tabloda ilgili adre
    se atilir
155.                    *total_word_count = *total_word_count + 1; //tablodaki kelime say
    isi guncellenir
156.                    hash_table[0].loadFactor =(float) *total_word_count / (float) M;
    //load factor hesaplanir
157.                    printf("%s kelimesi hash tablosuna eklendi\n",hash_table[adr].wor
    d); //bilgilendirme printleri atilir
158.                    printf("Load Factor : %.3f\n",hash_table[0].loadFactor);
159.                    //printf("count : %d\n",*total_word_count);
160.
161.                    return 1; //fonksiyondan cikilir
162.                }
163.            }
164.        }
165.
166.
```

```
167.
168.       //dosyadan veriyi satir satir okuyan fonksiyon
169.       int readInputFile(char *file_name, int *total_word_count, struct node *hash_t
    able) {
170.           FILE *inputFile; //input file 'ı açmaya yarayan file pointer
171.           char ch; //dosyadan karakterler okunup bu degiskene aktarilir(satir sayis
    ini hesaplamak icin)
172.           char *buffer = (char*) calloc(BUFFER_SIZE,sizeof(char)); //dosyadan alina
    n satirin tutuldugu buffer
173.           char *word  = (char*) calloc (WORD_SIZE,sizeof(char));//satirdan parcalan
    ip alinan kelime
174.           char *org_word = (char*) calloc (WORD_SIZE,sizeof(char));//orjinal kelime
    yi tutar
175.           int line_count = 0; //dosyadaki satir sayisini tutan degisken
176.           int i, j; //dongu degiskeni
177.           int isExist; //kelime hash table'da var mi yok mu donus degerini tutan de
    gisken
178.           //int wordCount = total_word_count;
179.
180.           if((inputFile = fopen(file_name,"r")) == NULL) {
181.               printf("Dosya okunmak icin acilamadi!\n");
182.               return 1;
183.           }
184.           else {
185.               //dosyadaki satir sayisini hesaplayan do-while bloğu
186.               do
187.               {
188.                   ch = fgetc(inputFile); //karakter oku
189.                   //new line ise line_count'u 1 arttır
190.                   if (ch == '\n') {
191.                       line_count++;
192.                   }
193.               } while (ch != EOF); //dosya sonuna gelene kadar
194.               rewind(inputFile); //dosyada basa don
195.               line_count++; //line count son haline guncellenir
196.
197.               //dosyadan veriler satir satir okunur ve isleme alinir
198.               for(i = 0; i < line_count; i++) {
199.                   fgets(buffer,BUFFER_SIZE * sizeof(char),inputFile); //satırı dosy
    adan buffer'a al
200.                   buffer[strlen(buffer)] = '\0'; //bufferin sinirini belirle
201.                   printf("--------------------------------------------------
    \n");
202.                   printf("Okunan satir : %s\n\n",buffer);
203.                   system("PAUSE");
204.                   printf("\n");
205.                   word = strtok(buffer, AYRAC); //satirdaki ilk kelime alinir
206.                   word[strlen(word)] = '\0'; //kelime sinirini belirle
207.                   strcpy(org_word,word); //alinan orjinal kelime org_word'de saklan
    ir
208.                   //case insensitive durum saglanmasi icin kelimedeki butun harfler
     kucuk harfe cevrilir
209.                   for(j = 0; j <strlen(word); j++) {
210.                       word[j] = tolower(word[j]);
211.                   }
212.       //          printf("word : %s\n",word);
213.       //          system("PAUSE");
214.                   isExist = insertTable(word,file_name,total_word_count,hash_table)
    ; //kelime hash tablosuna eklenmek uzere insertTable fonksiyonu cagirilir
215.                   //donus degeri kontrol edilir
216.                   if(isExist == -1) {
217.                       printf("%s kelimesi zaten tabloda mevcut!\n\n",org_word); //k
    elime hash tablosunda mevcutsa blgilendirme printi atilir
218.                   }
219.       //          else if(isExist == 1) {
220.       //              //printf("%s kelimesi eklendi!\n\n",org_word);
```

```
221.       //          }
222.
223.                   word = strtok(NULL, AYRAC); //satirdaki sonraki kelime alinir
224.                   //satirdaki diger kelimeler parcalara ayrilarak ayristirilir
225.                   while( word != NULL ) {
226.                       //satir sonuna gelindiyse new line karakter maskelenir
227.                       if(word[strlen(word)-1] == '\n') {
228.                           word[strlen(word)-1] = '\0';
229.                       }
230.                       strcpy(org_word,word); //alinan orjinal kelime org_word'de sa
    klanir
231.                       //case insensitive durum saglanmasi icin kelimedeki butun har
    fler kucuk harfe cevrilir
232.                       for(j = 0; j <strlen(word); j++) {
233.                           word[j] = tolower(word[j]);
234.                       }
235.                       word[strlen(word)] = '\0';  //kelime siniri belirlenir
236.       //              printf("word : %s, size %d\n",word,strlen(word));
237.       //              system("PAUSE");
238.                       isExist = insertTable(word,file_name,total_word_count,hash_ta
    ble); //kelime hash tablosuna eklenmek uzere insertTable fonksiyonu cagirilir
239.                       if(isExist == -1) {
240.                           printf("%s kelimesi zaten tabloda mevcut!\n\n",org_word);
     //kelime hash tablosunda mevcutsa blgilendirme printi atilir
241.                       }
242.       //              else if(isExist == 1) {
243.       //                  //printf("%s kelimesi eklendi!\n\n",org_word);
244.       //              }
245.                       word = strtok(NULL, AYRAC); //satirdaki diger kelimeler alini
    r
246.                   }
247.
248.               }
249.               fclose(inputFile); //dosya kapatilir
250.           }
251.           //free islemleri
252.           free(buffer);
253.           free(word);
254.           free(org_word);
255.           return 0; //fonksiyondan cikilir
256.       }
257.
258.       //icine verilen kelimeyi hash tablosunda arayan fonksiyon
259.       void searchHash (struct node *hash_table) {
260.           char *word = (char*) calloc(WORD_SIZE, sizeof(char)); //kullanicadan alin
    an ve tabloada aranacak olan kelime
261.           int cont = 1; //yeni arama olup olmayacagini kontrol eden degisken
262.           int i = 0; //kelimenin tablodaki adresi double probinge gore hesaplanirke
    n adim sayisini tutan degisken
263.           long long key; //kelimenin horner methoduna gore key degeri
264.           int adr; //kelimenin tablodaki adresi
265.           int j;  //dongu degiskeni
266.           while(cont) {
267.               printf("\nAramak istediginiz kelimeyi giriniz : \n");
268.               scanf("%s",word);   //aranmak istenen kelime kullanicidan alinir
269.               word[strlen(word)] = '\0';  //kelime siniri belirlenir
270.               //kelime case insensitive'lik saglanmasi icin kucuk harflere cevirili
    r
271.               for(j = 0; j <strlen(word); j++) {
272.                   word[j] = tolower(word[j]);
273.               }
274.       //      printf("kontrol!!! kelime : %s\n",word);
275.       //      system("pause");
276.               printf("\n");
277.               key = (long long) getHorner(word);  //horner sayisi alinir
278.
```

```
279.                //kelimenin tablodaki adresi double probing yontemiyle hesaplanir
280.                int h1_key = key % M;
281.                int h2_key = 1 + (key % MM);
282.                adr = (h1_key + (i * h2_key)) % M;
283.                i++;  //adim sayisi guncellenir
284.                //kelime tabloda mevcut degilse ilgili gozde bulunan struct'ın text_c
    ount degeri 0 demektir.
285.                if(hash_table[adr].text_count == 0) {
286.                    printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda tama
    mlanmistir.\n\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz.\n",i);
287.                    scanf("%d",&cont);
288.                }
289.                //kelimenin tabloda mevcut olma ihtimalinde
290.                else {
291.                    //kelime ile karsilasana kadar veya bos goz gorene kadar donen wh
    ile
292.                    while(hash_table[adr].text_count != 0 && strcmp(hash_table[adr].w
    ord, word) != 0) {
293.                        adr = (h1_key + (i * h2_key)) % M;  //adres degeri her adımda
     guncellenir
294.                        i++;    //adim sayisi guncellenir
295.                    }
296.                    //kelime ile karsilasildigi  icin while'dan cikilmissa kelime tab
    loda mevcuttur
297.                    if(strcmp(hash_table[adr].word, word) == 0) {
298.                        printf("Kelime tabloda mevcut!\nKelime toplamda %d adet dosya
    da gecmistir.\nKelimenin gectigi dosyalar : \n",hash_table[adr].text_count);
299.                        //kelimenin gectigi dosyalari ekrana yazdiran for
300.                        for(j = 0; j < hash_table[adr].text_count; j++) {
301.                            printf("%d. dosyanin ismi : %s\n",(j+1),hash_table[adr].t
    exts[j]);
302.                        }
303.                        printf("Arama islemi %d adimda tamamlanmistir.\n",i);
304.                        printf("\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz
    .\n");
305.                        scanf("%d",&cont);
306.                    }
307.                    //bos goze gelindigi icin whiledan cikilmissa kelime tabloda mevc
    ut degildir
308.                    else {
309.                        printf("Kelime tabloda mevcut degil!\nArama islemi %d adimda
    tamamlanmistir.\nYeni arama yapmak icin 1'e, cikmak icin 0'a basiniz.\n",i);
310.                        scanf("%d",&cont);  //yeni arama yapilacak mi bilgisi kullani
    cidan alinir
311.                    }
312.
313.                }
314.                //yeni aramalar icin word'u temizleyen for dongusu
315.                for(j = 0; j < WORD_SIZE; j++) {
316.                    word[j] = '\0';
317.                }
318.                i = 0; //yeni aramada adim sayisi 0'dan baslatilir
319.            }
320.            free(word);  //free islemi
321.        }
322.
323.
324.        int main() {
325.
326.            FILE *table_file;   //hashtable'ın okunacağı file icin file pointer
327.            char *file_name = (char*) calloc(TEXT_NAME_SIZE,sizeof(char));  //input d
    osyasinin adi
328.            struct node hashTable[M]; //hash table
329.            int total_word_count = 0;   //tablodaki kelime sayisini tutan degisken
330.            int control;    //secim kontrolu -
     dosyadan kelime okuma veya tablodan kelime arama
```

```
331.            int kontrol;    //input file'in okunup okunamadiğinin kontrolunu tutan de
    gisken
332.            //int i;
333.
334.        //  //tablo 0'dan olusturuldugunda ilklendirme foru
335.        //  for(i = 0; i < M; i++) {
336.        //      hashTable[i].text_count = 0;
337.        //      hashTable[i].loadFactor = 0;
338.        //  }
339.
340.            //hash tablosunun oldugu dosya acilir
341.            if ((table_file = fopen(HASH_TXT, "rb")) == NULL) {
342.                printf("Error opening file\n");
343.                return 1;
344.            }
345.            //hash tablosu dosyadan okunur
346.            else {
347.                fread(hashTable, sizeof(struct node) * M, 1, table_file);   //tabloyu
    oku
348.                fclose(table_file); //dosyayi kapat
349.            }
350.            total_word_count =  hashTable[0].loadFactor *(float) M;
351.            printf("Hash Table basariyla dosyadan okundu!\n\nTablo boyutu : %d\nTablo
    nun doluluk orani (load factor) : %.3f\n\n",M,hashTable[0].loadFactor);
352.
353.            //kullanicidan arama veya ekleme islemi icin istegi alinir
354.            printf("Hash Tablosunda arama yapmak icin 1'e, tabloya kelime eklemek ici
    n 2'ye basiniz.\n");
355.            scanf("%d",&control);
356.
357.            if(control == 1) {
358.                searchHash(hashTable);  //arama fonksiyonu cagirlir
359.
360.            }
361.            else if(control == 2) {
362.                printf("Input dosyasinin adini .txt uzantili olacak sekilde giriniz :
    ");
363.                scanf("%s",file_name);
364.                printf("\n");
365.                kontrol = readInputFile(file_name,&total_word_count,hashTable); //oku
    ma fonksiyonu cagirilir
366.                if(kontrol == 1) {
367.                    printf("Input dosyasi okunamadi!\n");
368.                    return 1; //dosya okunamazsa mainden cikilir
369.                }
370.        //      printf("\n\n\nMAINDEYIZ --
    total word: %d,load factor : %f\n\n\n",total_word_count,hashTable[0].loadFactor);
371.        //      system("pause");
372.            }
373.
374.
375.            //hash tablosunun oldugu dosya acilir
376.            if ( (table_file = fopen(HASH_TXT, "wb")) == NULL ) {
377.                printf("Error opening file\n");
378.                return 1;
379.            }
380.            //hash tablosu dosyaya yazilir
381.            else {
382.                    fwrite(hashTable, sizeof(struct node) * M, 1, table_file); //tabl
    o dosyaya yazilir
383.                    fclose(table_file); //dosya kapatilir
384.                    printf("\nTablo Dosyaya Basari ile Yazildi!\nTablonun doluluk ora
    ni (load factor) : %.3f\n",hashTable[0].loadFactor);
385.            }
386.
387.
```

```
388.            free(file_name); //free islemi
389.            return 0;   //end of main
390.        }
```