24.04.2020

YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM2512 VERİ YAPILARI VE ALGORİTMALAR 3.ÖDEV RAPORU

AHMET SAİD SAĞLAM

17011501

KONU

Find and Replace Uygulaması Hazırlama

Verilen ödevde, Boyer-Moore Horspool algoritması kullanılarak verilen bir kelimeyi bir metin içerisinde arayan ve bulduğu yerlerde gerekli değişikliği yapan bir algoritma tasarlanması ve kodunun C dilinde yazılması istenmiştir. Bununla birlikte algoritmanın kullanıcıya "Case Sensitive" seçeneği sunması istenmiştir. Böylece kullanıcı özelliği seçerek küçük büyük harf farkını aktif hale getirebilmelidir.

Çözüm için ilk olarak kullanıcıdan aranan kelimenin ne olduğu ve bulunması durumunda hangi ifade ile değiştirileceği bilgisi ile birlikte arama yapılacak metnin bulunduğu dosyanın ismi alınmıştır. Alınan bu bilgiler karakter dizilerinde saklanmıştır. Bunlarla birlikte case sensitive durumunun bilgisi de yine kullanıcıdan ilk aşamada alınmıştır.

İkinci aşamada text dosyasındaki metin bir karakter dizisine alınmış ve aranacak pattern için Horspool algoritmasının gereği olan Bad Match Table oluşturulmuştur. Bu aşamayla beraber pattern metin içinde aranabilir hale gelmiş bulunur.

Sonraki aşamada pattern metin içinde aranır ve bulunursa yeni ifade ile değiştirilir. Burada kod case sensitive durumuna göre küçük bir farkla çalışmaktadır. Eğer arama yapılırken case sensitive duruma göre yapılacaksa pattern aranır ve bulunduğu yerde değiştirilir. Diğer bir ihtimal olan case insensitive durumda ise bulunan patternlerin metnin tutulduğu dizide karşılık gelen indis numaraları kullanılarak başka bir fonksiyon içinde değiştirme işlemi yapılır. Bu farklılığın nedeni raporun ilerleyen kısımlarında açıklanmıştır.

Son bölümde, duruma göre değişen metin, alındığı text dosyasına yazdırılmıştır. Ekrana algoritmanın milisaniye cinsinden çalışma süresi, oluşan yeni metin, pattern'in kaç kez bulunup değiştirildiği ve dosyaya yeniden yazma bilgisi yazdırılarak program sonlandırılmıştır.

ÇÖZÜM

Kütüphane Eklenmesi ve Makro Atanması

```c
/*
@file
BLM2512 2019-2020 BAHAR ODEV-3
Bu programda Boyer-Moore Horspool Algoritması kullanılarak bir Find and Replace uygulaması tasarlanmıştır.

@author
İsim: Ahmet Said SAĞLAM
Öğrenci No: 17011501
Tarih: 24.04.2020
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>
#include <time.h>
#define MAXCHAR 4000 //Hafizaya alinip ustunde arama ve degistirme yapilabilecek maksimum karakter sayisi.
#define ASCII 256   //ascii karakter sayisi
```

Kodun ilk kısmında gerekli olabilecek kütüphaneler eklenmiş, daha sonra kod içerisinde gerekli bir makro atanmış ve programın geliştiricisi ve geliştirilip çalıştırıldığı ortam hakkında bilgiler verilmiştir.

MAXCHAR makrosu text dosyasından alınıp üstünde değişiklik yapılabilecek maksimum boyutlu diziyi tanımlarken kullanılmıştır. Bu sayede, istenilen durumda kolay bir şekilde değiştirilerek uzun boyutlu metinler için de hızlı çözümler üretilebilmiştir.

ASCII makrosuna ise Bad Match Table'ın uzunluğu olan 256 sayısı atanmıştır. Böylece program 256 farklı ascii karakter için Bad Match Table oluşturup arama yapabilir hale getirilmiştir.

Bad Match Table Oluşturulması

```
25  //bad match table olusturan fonksiyon
26  int *createBadMatch(char *search, int *bad_match) {
27      int i, j;
28      int search_size = strlen(search);   //bad-match table'da bulunan maksimum deger / stringde aranan kelimenin uzunlugu
29      int bad_line ;          //bad-match table olusturulurken kullanilan degisken
30
31      //bad match table'daki tum karakter indislerine, standart olarak aranan kelimenin uzunlugunu atayan dongu
32      for(i = 0; i < ASCII; i++){
33          bad_match[i] = search_size;
34      }
35
36      //bad match table'da zaten aranan kelimedeki karakterlerin kaydirma sayilarini duzenleyen dongu
37      for(i = 0; i < search_size-1; i++){
38          j = (int) search[i];
39          bad_line = search_size - 1 -i;
40          bad_match[j] = bad_line;
41      }
42      return bad_match;
43  }
```

createBadMatch fonksiyonu ilk olarak tüm ascii karakterlerin kaydırma sayısı olarak, aranacak pattern'in uzunluğunu ilgili karakterin ascii numarasına karşılık gelen bad_match dizisinin indisine atar. Sonrasında ise pattern'in içindeki karakterler için, pattern'in ilk karakterinden başlayarak, aranan pattern'in uzunluğu - pattern içindeki karakterin indisi - 1 kuralına göre kaydırma değerlerini düzenler. Pattern'in son karakteri bu döngüye dahil olmaz çünkü o karakter eğer pattern'de daha önceden geçmediyse standart olarak pattern'in uzunluğu son karakterin kaydırma sayısı olur.

Son adımda bad_match dizisi dışarıya döndürülür ve fonksiyon kapanır.

Case Sensitive Durum İçin Arama ve Değiştirme Yapılması

```
45  //horspool fonksiyonu case-sensitive durumda, text'te istenen pattern'i arar ve buldugu yerde pattern'i hedef ifade ile degistirir.
46  int horspool(char *text, char *pat, char *insert, int *badMatch) {
47      int eslesme_sayac = 0;      //eger pattern stringin icinde aranirken (karakter karakter karsilastirilirken) bir kac eslesmeden sonra b
48      int temp, k, kaydir;
49      int count = 0;                      //pattern'e kac defa rastlandigini tutan degisken
50      int patLength = strlen(pat);        //pattern'in uzunlugu
51      int insertLength = strlen(insert);  //pattern'in yerine yazilacak ifadenin uzunlugu
52      int fark, x;
53      int i = strlen(pat) - 1;    //i degiskeni  pattern'in son karakterini gosterecek sekilde ayarlanir.Boylece textte aramaya hangi karakt
54      int j = strlen(pat) - 1;    //j degiskeni  pattern'in son karakterini gosterecek sekilde ayarlanir.
55      fark = patLength - insertLength;
```

horspool fonksiyonu case sensitive durumda main fonksiyonu içinde çağırılarak arama ve değiştirme işlemini gerçekleştirir. Fonksiyon içine arama yapılacak pattern'i, pattern'in içinde bulunma olasılığı bulunan metni ve pattern'in bulunması durumunda yerine yazılacak yeni ifadeyi karakter dizisi şeklinde ve bunlarla birlikte bad match table'ı alır.

Fonksiyonun içinde ilk olarak fonksiyonda kullanılacak değişkenler tanımlanır.

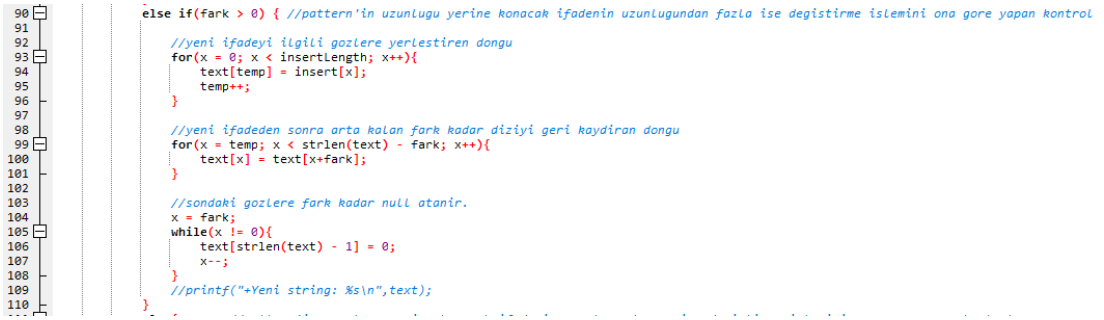
```
57      //text'in tum karakterlerini okumaya yardimci olan while dongusu
58      while(i < strlen(text)) {
59          temp = i;
60          //karakter eslesmesi durumunda bir sonraki karakteri kontrol ettiren ve eslesme sayaci tutan, pattern uzunlugu kadar donen while dongusu
61          while(text[temp] == pat[j] && j >= 0) {
62              temp--;
63              j--;
64              eslesme_sayac++;
65          }
66          if(j >= 0) {        // eger pattern bulunamadiysa girilen if kontrolu
67              k = text[temp];
68              kaydir = badMatch[k];
69              kaydir = kaydir - eslesme_sayac;
70              if(kaydir <= 0){
71                  i++;            //kaydirma degiskeni 0 veya daha kucuk bir deger alirsa text'te  bir adim ileri gidilir.
72              }
73              else{               //badmatch table'a gore text'te kaydirma yapilir
74                  i = i+kaydir;
75              }
76              j = strlen(pat) - 1;    //j degiskeni yeniden pattern'in son karakterini gosterecek sekilde ayarlanir.
77              eslesme_sayac = 0;      //eslesme sayaci 0'lanir.
78          }
```

Sonrasında bir while döngüsü yardımıyla i değişkeni kullanılarak metnin yani text'in ilk elemanlarından itibaren son elemanına kadar dönülür. İkinci while döngüsü yardımıyla kontrol başlar.

Eğer eşleşme olmazsa ikinci while döngüsüne hiç girilmez veya birkaç eşleşmeden sonra pattern tamamlanmadan bozulma olursa while'dan çıkılır ve hemen altındaki if koluna girilir. Burada horspool algoritmasına göre bad match table kullanılarak kaydırma yapılır ve yeniden kontrol başlar.

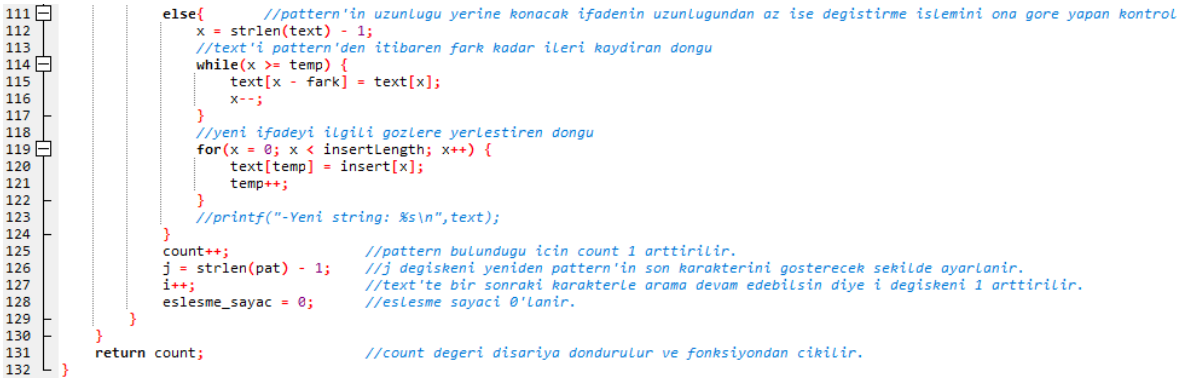
```
79          else {          //pattern bulunduysa girilen else kolu.
80              temp++;     //temp textte bulunan pattern'in ilk karakterinin indisini gosterecek sekilde ayarlanir.
81              if(fark == 0) {     //pattern ile yerine konacak ifade esit uzunluktaysa degistirme islemini ona gore yapan kontrol
82
83                  //yeni ifadeyi ilgili gozlere yerlestiren dongu
84                  for(x = 0; x < insertLength; x++) {
85                      text[temp] = insert[x];
86                      temp++;
87                  }
88                  //printf("0Yeni string: %s\n",text);
89              }
```

Eğer pattern bulunduysa kontrol while'ı olan ikinci while döngüsünün altındaki if kontrolünün else koluna girilir. Burada pattern ve yerine yazılacak ifadenin uzunluk farkına göre yerine yazma işlemi yapılır. İlk durumda, eğer pattern ve yerine yazılacak ifade eşit uzunluktaysa indisler kullanılarak değiştirme işlemi yapılır. Herhangi bir shift işlemine ihtiyaç olmadığından başvurulmaz.



```
90          else if(fark > 0) { //pattern'in uzunlugu yerine konacak ifadenin uzunlugundan fazla ise degistirme islemini ona gore yapan kontrol
91
92              //yeni ifadeyi ilgili gozlere yerlestiren dongu
93              for(x = 0; x < insertLength; x++){
94                  text[temp] = insert[x];
95                  temp++;
96              }
97
98              //yeni ifadeden sonra arta kalan fark kadar diziyi geri kaydiran dongu
99              for(x = temp; x < strlen(text) - fark; x++){
100                 text[x] = text[x+fark];
101             }
102
103             //sondaki gozlere fark kadar null atanir.
104             x = fark;
105             while(x != 0){
106                 text[strlen(text) - 1] = 0;
107                 x--;
108             }
109             //printf("+Yeni string: %s\n",text);
110         }
```

Eğer pattern'in uzunluğu, yerine konacak ifadeden uzun ise önce yeni ifade pattern'in üstüne yazılır. Sonrasında arta kalan boşluklardan sonraki ilk anlamlı karakterden itibaren text dizisi fark kadar geri kaydırılır. Son etapta ise dizinin en sonundan geriye doğru fark kadar elemanına null atanır. Bunun sayesinde sonda anlamsız karakter oluşmayacak ve dizinin boyu fark kadar kısalacaktır.



```
111         else{       //pattern'in uzunlugu yerine konacak ifadenin uzunlugundan az ise degistirme islemini ona gore yapan kontrol
112             x = strlen(text) - 1;
113             //text'i pattern'den itibaren fark kadar ileri kaydiran dongu
114             while(x >= temp) {
115                 text[x - fark] = text[x];
116                 x--;
117             }
118             //yeni ifadeyi ilgili gozlere yerlestiren dongu
119             for(x = 0; x < insertLength; x++) {
120                 text[temp] = insert[x];
121                 temp++;
122             }
123             //printf("-Yeni string: %s\n",text);
124         }
125         count++;                //pattern bulundugu icin count 1 arttirilir.
126         j = strlen(pat) - 1;    //j degiskeni yeniden pattern'in son karakterini gosterecek sekilde ayarlanir.
127         i++;                    //text'te bir sonraki karakterle arama devam edebilsin diye i degiskeni 1 arttirilir.
128         eslesme_sayac = 0;      //eslesme sayaci 0'lanir.
129     }
130 }
131 return count;                   //count degeri disariya dondurulur ve fonksiyondan cikilir.
132 }
```

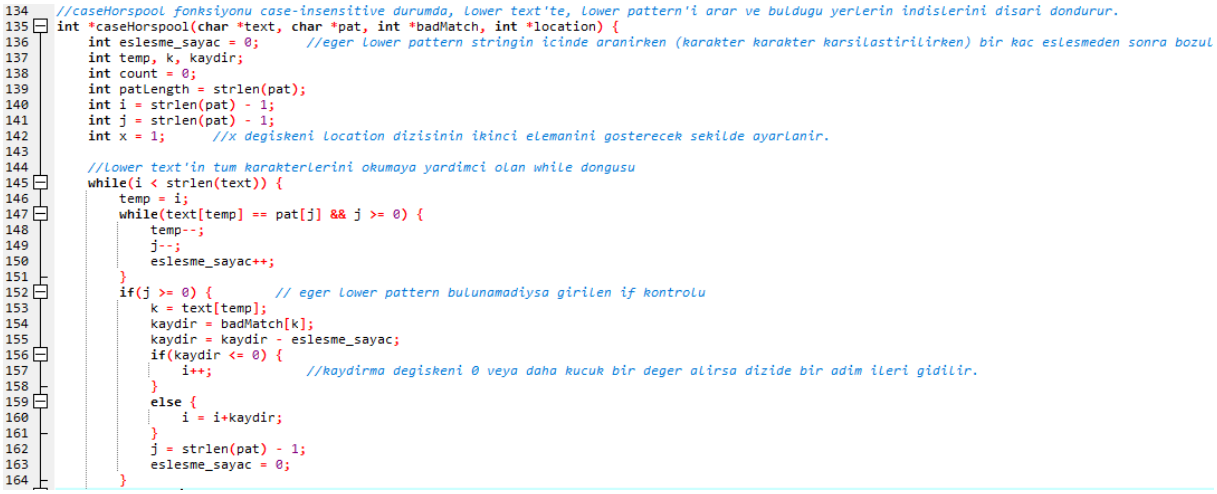
Son ihtimal ise pattern'in uzunluğunun yerine konacak ifadeden kısa olma ihtimalidir. Bu ihtimalde ise ilk olarak text dizisinin son elemandan pattern'in başladığı elemana kadar olan kısmı sondan başlayarak fark kadar ileri kaydırılır.

Kaydırma işlemi sonrasında pattern'in yerine yazılacak ifade pattern'in üstüne yazılır be else kolu kapanır. İf kontrollerinden sonra count değeri bir arttırılır. Böylece kaç kez değişim yapıldığı öğrenilmiş olur.

Yeni kontroller için değişkenler düzenlenir ve ilk while döngüsünün gereği olarak dizi sona kadar kontrol edilir.

En sonda ise count değeri dışarı döndürülür ve fonksiyondan çıkılır.

Case Insensitive Durum İçin Arama ve Değiştirme Yapılması

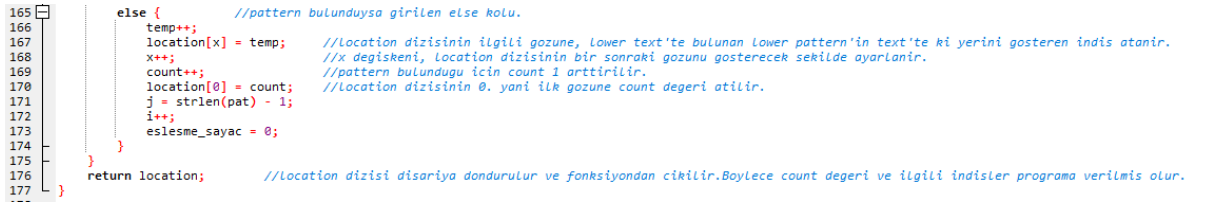


```
//caseHorspool fonksiyonu case-insensitive durumda, lower text'te, lower pattern'i arar ve buldugu yerlerin indislerini disari dondurur.
int *caseHorspool(char *text, char *pat, int *badMatch, int *location) {
    int eslesme_sayac = 0;      //eger lower pattern stringin icinde aranirken (karakter karakter karsilastirilirken) bir kac eslesmeden sonra bozul
    int temp, k, kaydir;
    int count = 0;
    int patLength = strlen(pat);
    int i = strlen(pat) - 1;
    int j = strlen(pat) - 1;
    int x = 1;      //x degiskeni location dizisinin ikinci elemanini gosterecek sekilde ayarlanir.

    //Lower text'in tum karakterlerini okumaya yardimci olan while dongusu
    while(i < strlen(text)) {
        temp = i;
        while(text[temp] == pat[j] && j >= 0) {
            temp--;
            j--;
            eslesme_sayac++;
        }
        if(j >= 0) {        // eger lower pattern bulunamadiysa girilen if kontrolu
            k = text[temp];
            kaydir = badMatch[k];
            kaydir = kaydir - eslesme_sayac;
            if(kaydir <= 0) {
                i++;            //kaydirma degiskeni 0 veya daha kucuk bir deger alirsa dizide bir adim ileri gidilir.
            }
            else {
                i = i+kaydir;
            }
            j = strlen(pat) - 1;
            eslesme_sayac = 0;
        }
```

Case insensitive durum için yapılan işlemler case sensitive durum için yapılanlarla hemen hemen aynıdır. Tek fark, değiştirme işleminin ayrı bir fonksiyonla yapılmasından kaynaklanır. Bunun nedeni ise case insensitive arama yapmak için orijinal text ve pattern'in bütün karakterlerinin küçük harfe çevrilmesi ve bu yeni metinlere göre arama yapılmasıdır.

Bu yeni metinlere göre yapılan arama sonucuna göre eğer pattern bulunursa, pattern'in veya pattern'lerin ilk karakterlerinin text içinde bulunduğu indisleri dışarıya döndürülür ve bu indisler sayesinde orijinal text'te değişim işlemi yapılır. Bu indisler ve kaç kez pattern bulunduğunu tutan count değeri location isimli dizi kullanılarak dışarı döndürülür.



```
        else {          //pattern bulunduysa girilen else kolu.
            temp++;
            location[x] = temp;     //location dizisinin ilgili gozune, lower text'te bulunan lower pattern'in text'te ki yerini gosteren indis atanir.
            x++;                    //x degiskeni, location dizisinin bir sonraki gozunu gosterecek sekilde ayarlanir.
            count++;                //pattern bulundugu icin count 1 arttirilir.
            location[0] = count;    //location dizisinin 0. yani ilk gozune count degeri atilir.
            j = strlen(pat) - 1;
            i++;
            eslesme_sayac = 0;
        }
    }
    return location;        //location dizisi disariya dondurulur ve fonksiyondan cikilir.Boylece count degeri ve ilgili indisler programa verilmis olur.
}
```

Daha önce bahsedildiği gibi eğer pattern bulunursa pattern'in ilk karakterinin indisi location dizisinin 1 numaralı gözüne atılır.

Sonraki bulunması ihtimal dahilinde olan pattern indisleri ise sırayla 2. Gözden itibaren location dizisine yerleşir. Bunun nedeni location dizisinin 0. gözünün count değerini tutmasıdır.

Son olarak location dizisi dışarı döndürülür ve fonksiyondan çıkılır.

```
179 //swap fonksiyonu case-insensitive durumda main'de cagirilir ve fonksiyona verilen text ve text'in ilgili gozundeki pattern'i degistirir.
180 //horsepool fonksiyonundaki degistirme blogunun yaptigi isin aynisini yapar.
181 void swap(int temp, int fark, int insertLength, char *text, char *pat, char *insert) {
182     int x;
183     if(fark == 0) {
184         for(x = 0; x < insertLength; x++) {
185             text[temp] = insert[x];
186             temp++;
187         }
188         //printf("0Yeni string: %s\n",text);
189     }
190     else if(fark > 0) {
191
192         //yeni ifadeyi ilgili gozlere yerlestiren dongu
193         for(x = 0; x < insertLength; x++){
194             text[temp] = insert[x];
195             temp++;
196         }
197
198         //yeni ifadeden sonra arta kalan fark kadar diziyi geri kaydiran dongu
199         for(x = temp; x < strlen(text) - fark; x++){
200             text[x] = text[x+fark];
201         }
202
203         //sondaki gozlere fark kadar null atanir.
204         x = fark;
205         while(x != 0){
206             text[strlen(text) - 1] = 0;
207             x--;
208         }
209         //printf("+Yeni string: %s\n",text);
210     }
```

```
211     else{
212         x = strlen(text) - 1;
213         while(x >= temp) {
214             text[x - fark] = text[x];
215             x--;
216         }
217         for(x = 0; x < insertLength; x++) {
218             text[temp] = insert[x];
219             temp++;
220         }
221         //printf("-Yeni string: %s\n",text);
222     }
223 }
```

Case insensitive durumda pattern'i değiştirme işlemi için swap fonksiyonu kullanılır. swap fonksiyonunun yaptığı değiştirme işlemi horspool fonksiyonu içinde yapılan değiştirme işlemi ile tamamen aynıdır. swap fonksiyonuna değiştirme işlemini yapabilmesi için değiştirmenin başlayacağı text dizisine ait gözün indisi temp değişkeninde verilir. Ayrıca swap fonksiyonuna orijinal text ve pattern verildiğinden case insensitive arama ve değiştirme işlemi tamamlanmış olur.

İnput Buffer Temizleme Fonksiyonu

```
226 //gets() fonksiyonu kullanılmadan önce eger scanf kullanıldıysa gets() duzgun calismayacagi icin input buffer'i temizlemmek icin gerekli fonksiyon
227 int clear_input_buffer(void) {
228     int ch;
229     while (((ch = getchar()) != EOF) && (ch != '\n'));
230     return ch;
231 }
```

clear_input_buffer fonksiyonu, kodun main fonksiyonu içinde 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.

Ana Fonksiyon Başlangıcı ve Tanımlamalar

```
int main(){
    FILE *fp;                       //file pointer
    int i, j, z;                    //dongulerde ve kontrollerde kullanilacak degiskenler
    int case_sensitive;             //case senstive durumunu kontrol eden degisken
    int bad_match[ASCII];           //tum ascii karakterlere gore deger tutan Bad-Match Table
    int *badMatch;                  //Bad-Match Table'ı işaret eden pointer
    int count;                      //pattern'in kac kere bulunup (degistirildigini) tutan degisken.
    int *location;                  //lokasyon dizisini işaret eden pointer
    int lokasyon[MAXCHAR];          //case insensitive durumunda count degerini ve text karakter dizisi uzerinde degisiklik yapilacak indis numaralarini tutan dizi
    char text[MAXCHAR];             //dosyadan alinan, kelimenin icinde aranacagi stringi tutan karakter dizisi
    char lowerText[MAXCHAR];        //dosyadan alinan text'in butun karakterlerini sirasiyla lowercase halde tutan karakter dizisi
    char filename[MAXCHAR];         //dosya ismini tutan karakter dizisi
    char search[MAXCHAR];           //hangi kelimenin aranacagini tutan karakter dizisi
    char lowerSearch[MAXCHAR];      //kullanicidan alinan pattern'in butun karakterlerini sirasiyla lowercase halde tutan karakter dizisi
    char insert[MAXCHAR];           //text'in icine uygun durumda hangi ifadenin yazilacagini tutan karakter dizisi
    int patLength;                  //pattern'in uzunlugu
    int insertLength;               //pattern'in yerine eklenecek ifadenin uzunlugu.
    int fark,size;                  //fark = pattern ve yeni ifadenin uzunluk farki, size = dosyadan alinacak fadenin uzunlugu
    clock_t start, end;             //sure baslangic ve bitisi
    double time_spent;              //toplam gecen sure(ms)

    lokasyon[0] = 0;        //case-insensitive durumda count degeri ilk an icin 0 olarak ayarlanir.
    //kullanilacak diziler ilk durum icin 0'lanir.
    for(i = 0; i < MAXCHAR; i++){
        text[i] = 0;
        search[i] = 0;
        lowerText[i] = 0;
        lowerSearch[i] = 0;
        insert[i] = 0;
    }
```

main fonksiyonunun içinde öncelikle kullanılacak değişkenlerin, dizilerin ve pointerların tanımı yapılır. Ardından kullanılacak diziler ilk durumda sıfırlanır ve program akışı devam eder.

Kullanıcıdan Gerekli Verilerin Alınması

```
    printf("Aramayi Case Insensitive (Buyuk-Kucuk harf duyarsiz) olarak yapmak isterseniz 1'e,\nCase Sensitive (Buyuk-Kucuk harf duyarli)
    scanf("%d",&case_sensitive);
    printf("Aramak istediginiz ifadeyi giriniz.\n");
    clear_input_buffer();
    gets(search);
    printf("Yerine yazmak istediginiz ifadeyi giriniz.\n");
    gets(insert);
    patLength = strlen(search);
    insertLength = strlen(insert);
    fark = patLength - insertLength;

    printf("Lutfen acmak istediginiz dosyanin ismini uzantisiyla birlikte giriniz.\n");
    scanf("%s", filename);
```

Programın devamında kullanıcıdan aramanın case sensitive olup olmayacağının bilgisi, aranacak ifade, yerine yazılacak ifade ve arama yapılacak metnin bulunduğu dosyanın ismi istenir. Bunlarla beraber aranacak ifade ile yerine yazılacak ifadenin uzunluğu ve ikisinin farkı hesaplanır.

Metnin Dosyadan Alınması ve Case Insensitive Durumun Gerçeklenmesi

```
278     if((fp = fopen(filename,"r")) == NULL) {
279         printf("Dosya okunmak icin acilamadi!\n");
280         return 0;
281     }
282     else {
283         if(case_sensitive == 1) {   //arama case-insensitive mi kontrol eden if statement
284
285             //aranacak kelimenin tum karakterleri kucuk karakter olacak sekilde kelimeyi yeniden duzenleyen dongu
286             for(i = 0; i < strlen(search); i++){
287                 lowerSearch[i] = tolower(search[i]);
288             }
289             badMatch = createBadMatch(lowerSearch,bad_match);   //badmatch table olusturulur
290             fseek(fp, 0, SEEK_END); // seek to end of file
291             size = ftell(fp); // get current file pointer
292             fseek(fp, 0, SEEK_SET); // seek back to beginning of file
293             fread(text,size,1,fp); //dosyadaki text text dizisine alinir.
294             printf("Text: %s\n",text);
295
296             //dosyadan alinan text'in tum karakterleri kucuk karakter olacak sekilde text'i duzenleyen dongu
297             for(i = 0; i < strlen(text); i++){
298                 lowerText[i] = tolower(text[i]);
299             }
```

Program sonraki aşamada kullanıcıdan alınan dosya ismi ile dosyayı okumak için açmaya çalışır ve bir hata oluşursa uyarı vererek programı sonlandırır. Dosya açılırsa ve kullanıcı case insensitive arama yapmak istemişse öncelikle aranacak pattern küçük harflere çevrilerek lowerSearch dizisinde tutulur. Sonrasında bu lowerSearch için bad match table oluşturulur ve dosyadan metin fread() ile okunur. Burada fread() kullanılmasının sebebi metin eğer birden fazla satırdan oluşuyorsa satırların hepsi ile birlikte bütün metni tek seferde okuyabilme gerekliliğindendir. Devamında dosyadan okunan metin önce ekrana yazdırılır daha sonra ise küçük harflere çevrilerek lowerText dizisinde tutulur.

```
303             start = clock(); //sure hesaplama baslangici
304             location = caseHorspool(lowerText,lowerSearch,badMatch,lokasyon); //pattern text'te aranir, count ve bulunan patternlerin yeri dondurulu
305             if(location[0] == 0) {      //count 0 ise text'te aranan pattern mevcut degildir.
306                 printf("Eslesme yok!\n");
307                 return 0;
308             }
```

Bu aşamadan itibaren artık algoritma çalışmaya başlayacağı için süre sayımı başlatılır. Sonrasında caseHorspool fonksiyonu çağırılır ve ondan dönen location dizisi alınır. Böylece lowerText'te lowerSearch(küçük harfli pattern) aranmış olur. Bu dizinin 0. yani ilk elemanı toplam pattern sayısını tutan count değerini temsil ettiğinden dizinin 0. elemanına ait değer 0 mı diye kontrol edilir. Eğer değer 0 ise text'te pattern'e hiç rastlanmamış demektir. Bu durumda program kullanıcıya uyarı verir ve sonlanır.

```
309         else {
310             j = 0;  //count-control
311             i = 1;  //location dizisinin ilgili gozune erisen indis numarasi.Boylece text'te hangi indisde degisim yapilacagi ogrenilir.
312 
313             //count kadar donerek swap islemi yaptiran while dongusu
314             while (j < location[0]) {
315                 if(j == 0){
316                     swap(location[i],fark,insertLength,text,search,insert);
317                     j++;
318                     i++;
319                 }
320                 else {
321                     //patternlerin baslangic indislerini guncelleyen for dongusu
322                     for(z = i; z <= location[0]; z++){
323                         location[z] = location[z] - fark;   //ilk degisimden sonra text dizisinde diger bulunan patternlerin yeri fark kadar kayac
324                     }
325                     swap(location[i],fark,insertLength,text,search,insert);
326                     j++;
327                     i++;
328                 }
329             }
330             end = clock(); //sure hesaplama bitisi.
331             time_spent = (double) (end - start);
332             printf("Son Text: %s\nDegisim sayisi: %d\nGecen zaman: %f ms\n",text,location[0],time_spent);
333         }
334         fclose(fp); //dosyayi okumak icin acildigindan kapatir.
```

Eğer değer 0'dan farklı ise pattern text'in içinde var demektir ve pattern'in bulunduğu indis veya indisler location dizisinde mevcut anlamına gelir. Bu durumda count değeri kadar swap fonksiyonu çağırılır ve orijinal text'te değişiklikler yapılır. Burada önemli bir ayrıntı bulunmaktadır. İlk değişiklikten sonra, eğer pattern ve yerine konacak yeni ifade arasındaki uzunluk farkı 0'dan farklı ise diğer bulunan patternlerin indis numaraları fark kadar kayacaktır. Bu sorunu çözebilmek için for döngüsüyle beraber ilgili indis numaraları farka bağlı olarak güncellenir ve ikinci ve daha sonraki her değişiklikten önce bu işlem tekrarlanır.

Değiştirme işlemlerinin de bitişinin ardından süre hesaplaması durdurulur ve geçen süre milisaniye cinsinden hesaplanır. Ardından oluşan son text, toplam değişim sayısı ve geçen zaman ekrana yazdırılarak dosya kapatılır.

```
336             //dosyayi yazmak icin yeniden acar ve yeni text'i icine yazar.
337             if((fp = fopen(filename,"w")) == NULL) {
338                 printf("Dosya yazmak icin acilamadi!\n");
339                 return 0;
340             }
341             else {
342                 fputs(text,fp);
343                 fclose(fp);
344                 printf("Dosya Icerigi Guncellendi!\n");
345             }
346         }
```

Case insensitive durum için son aşamada dosya yeniden açılır ve oluşan yeni text içine yazılır.

Metnin Dosyadan Alınması ve Case Sensitive Durumun Gerçeklenmesi

```
347         else {  //case sensitive durum.
348             badMatch = createBadMatch(search,bad_match); //bad-match table olusturulur
349             fseek(fp, 0, SEEK_END); // seek to end of file
350             size = ftell(fp); // get current file pointer
351             fseek(fp, 0, SEEK_SET); // seek back to beginning of file
352             fread(text,size,1,fp);
353             printf("Text: %s\n",text);
354             start = clock();    //sure hesaplama baslangici
355             count = horspool(text, search, insert, badMatch);   //text'te pattern bulunduysa ilgili degisiklik yapilir ve count degeri disari donulur.
356             if(count == 0){
357                 printf("Eslesme yok!\n");
358                 return 0;
359             }
360             end = clock();  //sure hesaplama bitisi
361             time_spent = (double) (end - start);
362             //printf("Count: %d\n",count);
363             printf("Son Text: %s\nDegisim sayisi: %d\nGecen zaman: %f ms\n",text,count,time_spent);
364             fclose(fp); //dosyayi okumak icin acildigindan kapatir.
```

Case sensitive durumda ilk olarak kullanıcıdan alınan pattern için bad match table oluşturulur ve hemen ardından metin text dosyasından okunur ve ekrana yazdırılır. Bu aşamadan itibaren algoritma çalışacağı için süre sayımı başlatılır ve horspool fonksiyonu çağırılır. Böylece arama ve varsa değiştirme işlemi tek seferde bitirilmiş olur. horspool fonksiyonundan dönen count değeri kontrol edilir ve eğer değer 0 ise kullanıcıya uyarı verilerek program sonlanır. Ardından algoritmanın çalışması bittiğinden süre sayımı durdurulur ve geçen süre milisaniye cinsinden hesaplanır.

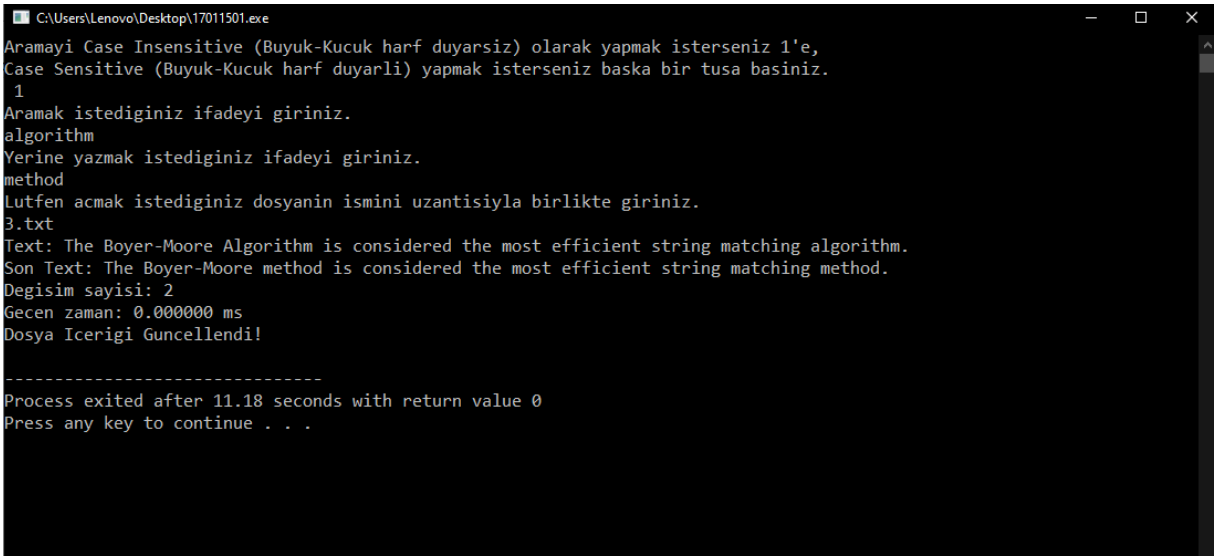
Oluşan son text, toplam değişim sayısı ve geçen zaman ekrana yazdırılarak dosya kapatılır.

```
366                 //dosyayi yeniden acar ve yeni text'i icine yazar.
367                 if((fp = fopen(filename,"w")) == NULL) {
368                     printf("Dosya yazmak icin acilamadi!\n");
369                     return 0;
370                 }
371                 else {
372                     fputs(text,fp);
373                     fclose(fp);
374                     printf("Dosya Icerigi Guncellendi!\n");
375                 }
376             }
377         }
378         return 0;
379     }
```

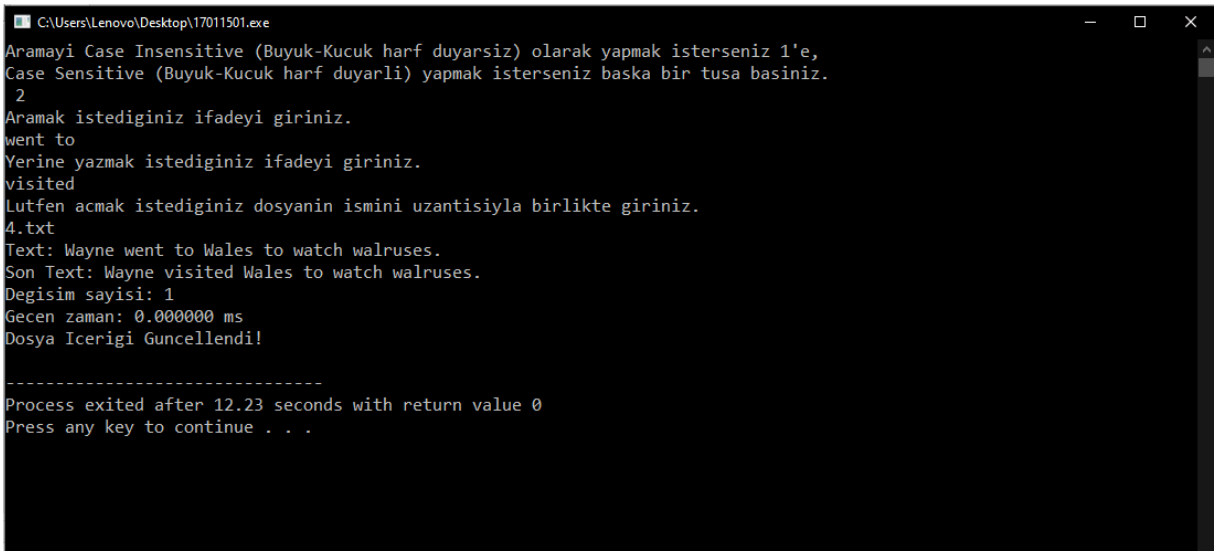
Programın son kısmında case sensitive durum için dosya yeniden açılır ve oluşan yeni text içine yazılır.

Ardından program sonlanmış olur.

PROGRAM ÇIKTILARI



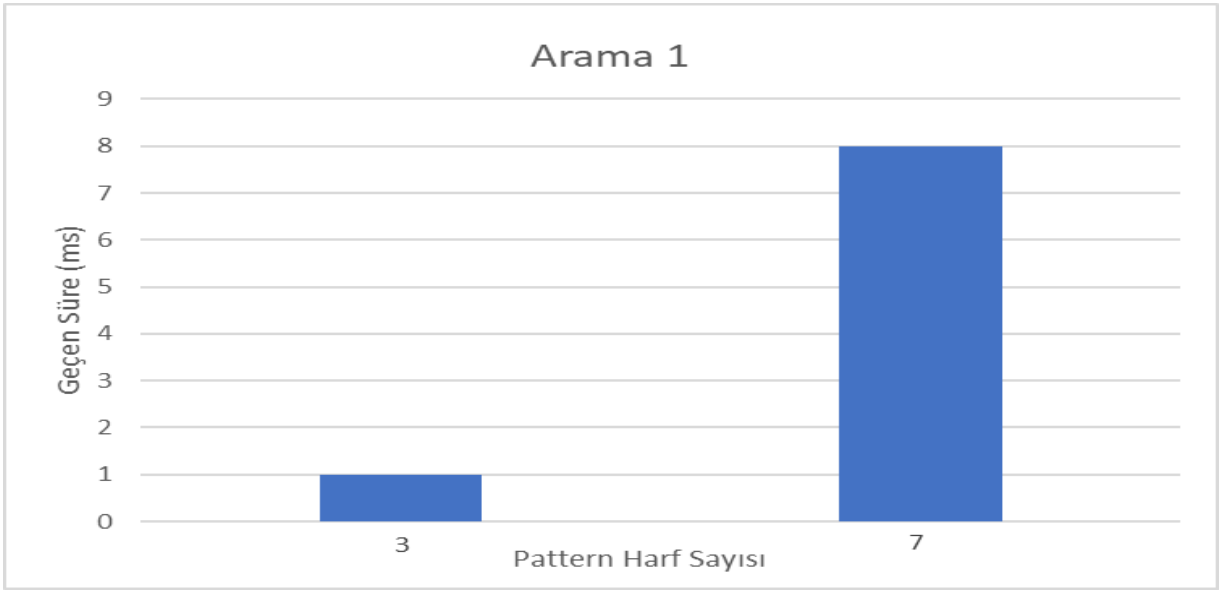
Case Insensitive Arama ve Değiştirme



Case Sensitive Arama ve Değiştirme

SÜRE ANALİZİ

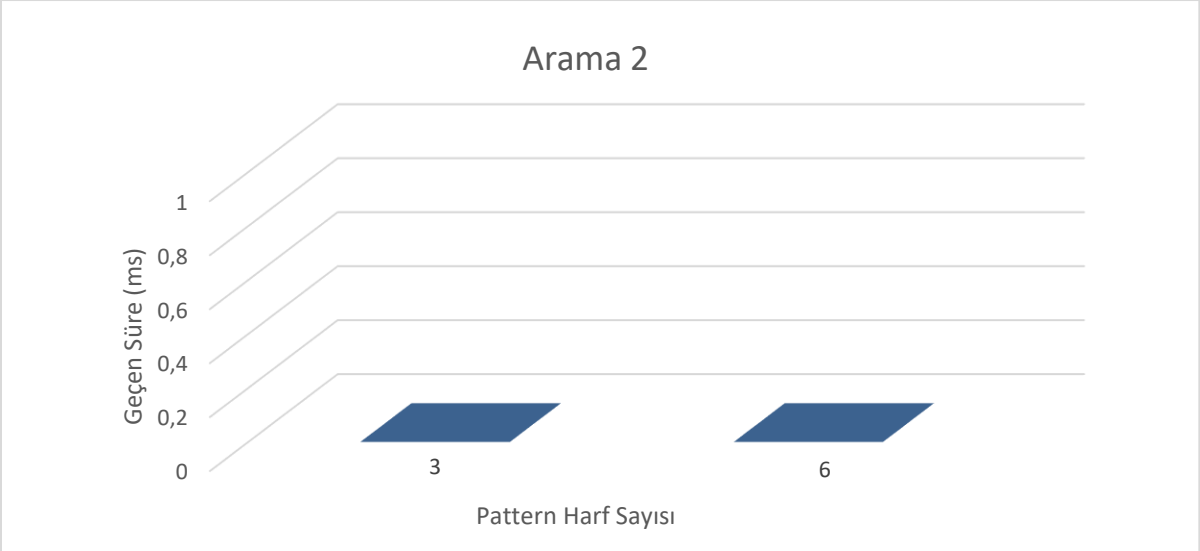
Örnek 1:



Yukarıda görülen tabloda 2500 karaktere sahip metinde yapılan arama ve değiştirme sonuçları görülmektedir. Her iki örnek için de 24 adet pattern bulunmuş ve değiştirilmiştir. Değiştirilen ifadeler aranan pattern'den birkaç karakter daha uzundur.

Aynı işlem 8 karaktere sahip başka bir pattern için tekrarlandığında, 2500 karakterlik metinde sadece 1 kez bulunmuş ve değiştirilmiştir. Ancak toplam geçen süre 0 ms olarak hesaplanmıştır.

Örnek 2:



Yukarıda görülen tabloda 400 karaktere sahip metinde yapılan arama ve değiştirme sonuçları görülmektedir. Her iki örnek için de 12 adet pattern bulunmuş ve değiştirilmiştir. Değiştirilen ifadeler aranan pattern'den birkaç karakter daha uzundur.

Aynı işlem 62 karaktere sahip bir cümle olan başka bir pattern için tekrarlandığında, 400 karakterlik metinde 1 kez bulunmuş ve değiştirilmiştir. Toplam geçen süre yine 0 ms olarak hesaplanmıştır.

Bu iki örnek bize geçen süreyi en çok etkileyen faktörün arama yapılan metnin uzunluğu olduğunu göstermektedir. Aynı metinde yapılan farklı aramalarda ise aranan pattern'in uzunluğu ve kaç kez tekrar ettiği süreyi ciddi şekilde etkilemektedir. Geçen süre açısından en kötü durum, uzun bir metinde aranan uzun bir pattern'in çok kez tekrar etmesi iken en iyi durum ise, kısa bir metinde aranan kısa bir pattern'in az tekrar sayısına sahip olmasıdır.

SOURCE CODE

```
1.  /*
2.  @file
3.  BLM2512 2019-2020 BAHAR ODEV-3
4.  Bu programda Boyer-
    Moore Horspool Algoritması kullanılarak bir Find and Replace uygulaması tasarlanmışt
    ır.
5.
6.  @author
7.  İsim: Ahmet Said SAĞLAM
8.  Öğrenci No: 17011501
9.  Tarih: 24.04.2020
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. #include <stdio.h>
16. #include <stdlib.h>
17. #include <string.h>
18. #include <conio.h>
19. #include <stdbool.h>
20. #include <ctype.h>
21. #include <time.h>
22. #define MAXCHAR 4000 //Hafizaya alinip ustunde arama ve degistirme yapilabilecek mak
    simum karakter sayisi.
23. #define ASCII 256   //ascii karakter sayisi
24.
25. //bad match table olusturan fonksiyon
26. int *createBadMatch(char *search, int *bad_match) {
27.     int i, j;
28.     int search_size = strlen(search);   //bad-
    match table'da bulunan maksimum deger / stringde aranan kelimenin uzunlugu
29.     int bad_line ;          //bad-match table olusturulurken kullanilan degisken
30.
31.     //bad match table'daki tum karakter indislerine, standart olarak aranan kelimeni
    n uzunlugunu atayan dongu
32.     for(i = 0; i < ASCII; i++){
33.         bad_match[i] = search_size;
34.     }
35.
36.     //bad match table'da zaten aranan kelimedeki karakterlerin kaydirma sayilarini d
    uzenleyen dongu
37.     for(i = 0; i < search_size-1; i++){
38.         j = (int) search[i];
39.         bad_line = search_size - 1 -i;
40.         bad_match[j] = bad_line;
41.     }
42.     return bad_match;
43. }
44.
45. //horspool fonksiyonu case-
    sensitive durumda, text'te istenen pattern'i arar ve buldugu yerde pattern'i hedef i
    fade ile degistirir.
46. int horspool(char *text, char *pat, char *insert, int *badMatch) {
47.     int eslesme_sayac = 0;      //eger pattern stringin icinde aranirken (karakter k
    arakter karsilastirilirken) bir kac eslesmeden sonra bozulma olursa bad match table'
    daki degerleri eslesme sayisina gore duzenleyecek olan degisken
48.     int temp, k, kaydir;
49.     int count = 0;                      //pattern'e kac defa rastlandigini tutan deg
    isken
50.     int patLength = strlen(pat);        //pattern'in uzunlugu
51.     int insertLength = strlen(insert);  //pattern'in yerine yazilacak ifadenin uzunl
    ugu
```

```
52.     int fark, x;
53.     int i = strlen(pat) -
     1;    //i degiskeni  pattern'in son karakterini gosterecek sekilde ayarlanir.Boylec
    e textte aramaya hangi karakterden baslanacagi belirlenmis olur.
54.     int j = strlen(pat) -
     1;    //j degiskeni  pattern'in son karakterini gosterecek sekilde ayarlanir.
55.     fark = patLength - insertLength;
56.
57.     //text'in tum karakterlerini okumaya yardimci olan while dongusu
58.     while(i < strlen(text)) {
59.         temp = i;
60.         //karakter eslesmesi durumunda bir sonraki karakteri kontrol ettiren ve esle
    sme sayaci tutan, pattern uzunlugu kadar donen while dongusu
61.         while(text[temp] == pat[j] && j >= 0) {
62.             temp--;
63.             j--;
64.             eslesme_sayac++;
65.         }
66.         if(j >= 0) {     // eger pattern bulunamadiysa girilen if kontrolu
67.             k = text[temp];
68.             kaydir = badMatch[k];
69.             kaydir = kaydir - eslesme_sayac;
70.             if(kaydir <= 0){
71.                 i++;            //kaydirma degiskeni 0 veya daha kucuk bir deger ali
    rsa text'te  bir adim ileri gidilir.
72.             }
73.             else{               //badmatch table'a gore text'te kaydirma yapilir
74.                 i = i+kaydir;
75.             }
76.             j = strlen(pat) -
     1;    //j degiskeni yeniden pattern'in son karakterini gosterecek sekilde ayarlanir
    .
77.             eslesme_sayac = 0;      //eslesme sayaci 0'lanir.
78.         }
79.         else {          //pattern bulunduysa girilen else kolu.
80.             temp++;     //temp textte bulunan pattern'in ilk karakterinin indisini g
    osterecek sekilde ayarlanir.
81.             if(fark == 0) {     //pattern ile yerine konacak ifade esit uzunluktaysa
     degistirme islemini ona gore yapan kontrol
82.
83.                 //yeni ifadeyi ilgili gozlere yerlestiren dongu
84.                 for(x = 0; x < insertLength; x++) {
85.                     text[temp] = insert[x];
86.                     temp++;
87.                 }
88.                 //printf("0Yeni string: %s\n",text);
89.             }
90.             else if(fark > 0) {  //pattern'in uzunlugu yerine konacak ifadenin uzunl
    ugundan fazla ise degistirme islemini ona gore yapan kontrol
91.
92.                 //yeni ifadeyi ilgili gozlere yerlestiren dongu
93.                 for(x = 0; x < insertLength; x++){
94.                     text[temp] = insert[x];
95.                     temp++;
96.                 }
97.
98.                 //yeni ifadeden sonra arta kalan fark kadar diziyi geri kaydiran don
    gu
99.                 for(x = temp; x < strlen(text) - fark; x++){
100.                    text[x] = text[x+fark];
101.                }
102.
103.                //sondaki gozlere fark kadar null atanir.
104.                x = fark;
105.                while(x != 0){
106.                    text[strlen(text) - 1] = 0;
```

```
                        x--;
                    }
                    //printf("+Yeni string: %s\n",text);
                }
                else{       //pattern'in uzunlugu yerine konacak ifadenin uzunlug
undan az ise degistirme islemini ona gore yapan kontrol
                    x = strlen(text) - 1;
                    //text'i pattern'den itibaren fark kadar ileri kaydiran dongu

                    while(x >= temp) {
                        text[x - fark] = text[x];
                        x--;
                    }
                    //yeni ifadeyi ilgili gozlere yerlestiren dongu
                    for(x = 0; x < insertLength; x++) {
                        text[temp] = insert[x];
                        temp++;
                    }
                    //printf("-Yeni string: %s\n",text);
                }
                count++;                //pattern bulundugu icin count 1 arttiril
ir.
                j = strlen(pat) -
 1;    //j degiskeni yeniden pattern'in son karakterini gosterecek sekilde ayarlanir
.
                i++;                    //text'te bir sonraki karakterle arama de
vam edebilsin diye i degiskeni 1 arttirilir.
                eslesme_sayac = 0;      //eslesme sayaci 0'lanir.
            }
        }
        return count;                   //count degeri disariya dondurulur ve fon
ksiyondan cikilir.
    }

    //caseHorspool fonksiyonu case-
insensitive durumda, lower text'te, lower pattern'i arar ve buldugu yerlerin indisle
rini disari dondurur.
    int *caseHorspool(char *text, char *pat, int *badMatch, int *location) {
        int eslesme_sayac = 0;      //eger lower pattern stringin icinde aranirke
n (karakter karakter karsilastirilirken) bir kac eslesmeden sonra bozulma olursa bad
 match table'daki degerleri eslesme sayisina gore duzenleyecek olan degisken
        int temp, k, kaydir;
        int count = 0;
        int patLength = strlen(pat);
        int i = strlen(pat) - 1;
        int j = strlen(pat) - 1;
        int x = 1;      //x degiskeni location dizisinin ikinci elemanini gostere
cek sekilde ayarlanir.

        //lower text'in tum karakterlerini okumaya yardimci olan while dongusu
        while(i < strlen(text)) {
            temp = i;
            while(text[temp] == pat[j] && j >= 0) {
                temp--;
                j--;
                eslesme_sayac++;
            }
            if(j >= 0) {     // eger lower pattern bulunamadiysa girilen if kontr
olu
                k = text[temp];
                kaydir = badMatch[k];
                kaydir = kaydir - eslesme_sayac;
                if(kaydir <= 0) {
                    i++;            //kaydirma degiskeni 0 veya daha kucuk bir de
ger alirsa dizide bir adim ileri gidilir.
                }
```

```
159.                    else {
160.                        i = i+kaydir;
161.                    }
162.                    j = strlen(pat) - 1;
163.                    eslesme_sayac = 0;
164.                }
165.                else {          //pattern bulunduysa girilen else kolu.
166.                    temp++;
167.                    location[x] = temp;     //location dizisinin ilgili gozune, lower
     text'te bulunan lower pattern'in text'te ki yerini gosteren indis atanir.
168.                    x++;                    //x degiskeni, location dizisinin bir son
    raki gozunu gosterecek sekilde ayarlanir.
169.                    count++;                //pattern bulundugu icin count 1 arttiril
    ir.
170.                    location[0] = count;    //location dizisinin 0. yani ilk gozune c
    ount degeri atilir.
171.                    j = strlen(pat) - 1;
172.                    i++;
173.                    eslesme_sayac = 0;
174.                }
175.            }
176.            return location;        //location dizisi disariya dondurulur ve fonksiyo
    ndan cikilir.Boylece count degeri ve ilgili indisler programa verilmis olur.
177.        }
178.
179.        //swap fonksiyonu case-
    insensitive durumda main'de cagirilir ve fonksiyona verilen text ve text'in ilgili g
    ozundeki pattern'i degistirir.
180.        //horsepool fonksiyonundaki degistirme blogunun yaptigi isin aynisini yapar.

181.        void swap(int temp, int fark, int insertLength, char *text, char *pat, char *
    insert) {
182.            int x;
183.            if(fark == 0) {
184.                for(x = 0; x < insertLength; x++) {
185.                    text[temp] = insert[x];
186.                    temp++;
187.                }
188.                //printf("0Yeni string: %s\n",text);
189.            }
190.            else if(fark > 0) {
191.
192.                //yeni ifadeyi ilgili gozlere yerlestiren dongu
193.                for(x = 0; x < insertLength; x++){
194.                    text[temp] = insert[x];
195.                    temp++;
196.                }
197.
198.                //yeni ifadeden sonra arta kalan fark kadar diziyi geri kaydiran dong
    u
199.                for(x = temp; x < strlen(text) - fark; x++){
200.                    text[x] = text[x+fark];
201.                }
202.
203.                //sondaki gozlere fark kadar null atanir.
204.                x = fark;
205.                while(x != 0){
206.                    text[strlen(text) - 1] = 0;
207.                    x--;
208.                }
209.                //printf("+Yeni string: %s\n",text);
210.            }
211.            else{
212.                x = strlen(text) - 1;
213.                while(x >= temp) {
214.                    text[x - fark] = text[x];
```

```
                    x--;
                }
                for(x = 0; x < insertLength; x++) {
                    text[temp] = insert[x];
                    temp++;
                }
                //printf("-Yeni string: %s\n",text);
            }
        }


        //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;
        }


        int main(){
            FILE *fp;                       //file pointer
            int i, j, z;                    //dongulerde ve kontrollerde kullanilacak degiskenler
            int case_sensitive;             //case senstive durumunu kontrol eden degisken
            int bad_match[ASCII];           //tum ascii karakterlere gore deger tutan Bad-Match Table
            int *badMatch;                  //Bad-Match Table'ı işaret eden pointer
            int count;                      //pattern'in kac kere bulunup (degistirildigini) tutan degisken.
            int *location;                  //lokasyon dizisini isaret eden pointer
            int lokasyon[MAXCHAR];          //case insensitive durumunda count degerini ve text karakter dizisi uzerinde degisiklik yapilacak indis numaralarini tutan dizi
            char text[MAXCHAR];             //dosyadan alinan, kelimenin icinde aranacagi stringi tutan karakter dizisi
            char lowerText[MAXCHAR];        //dosyadan alinan text'in butun karakterlerini sirasiyla lowercase halde tutan karakter dizisi
            char filename[MAXCHAR];         //dosya ismini tutan karakter dizisi
            char search[MAXCHAR];           //hangi kelimenin aranacagini tutan karakter dizisi
            char lowerSearch[MAXCHAR];      //kullanicidan alinan pattern'in butun karakterlerini sirasiyla lowercase halde tutan karakter dizisi
            char insert[MAXCHAR];           //text'in icine uygun durumda hangi ifadenin yazilacagini tutan karakter dizisi
            int patLength;                  //pattern'in uzunlugu
            int insertLength;               //pattern'in yerine eklenecek ifadenin uzunlugu.
            int fark,size;                  //fark = pattern ve yeni ifadeinin uzunluk farki, size = dosyadan alinacak fadenin uzunlugu
            clock_t start, end;             //sure baslangic ve bitisi
            double time_spent;              //toplam gecen sure(ms)

            lokasyon[0] = 0;        //case-insensitive durumda count degeri ilk an icin 0 olarak ayarlanir.
            //kullanilacak diziler ilk durum icin 0'lanir.
            for(i = 0; i < MAXCHAR; i++){
                text[i] = 0;
                search[i] = 0;
                lowerText[i] = 0;
                lowerSearch[i] = 0;
                insert[i] = 0;
            }

```

```
265.            printf("Aramayi Case Insensitive (Buyuk-
    Kucuk harf duyarsiz) olarak yapmak isterseniz 1'e,\nCase Sensitive (Buyuk-
    Kucuk harf duyarli) yapmak isterseniz baska bir tusa basiniz.\n ");
266.            scanf("%d",&case_sensitive);
267.            printf("Aramak istediginiz ifadeyi giriniz.\n");
268.            clear_input_buffer();
269.            gets(search);
270.            printf("Yerine yazmak istediginiz ifadeyi giriniz.\n");
271.            gets(insert);
272.            patLength = strlen(search);
273.            insertLength = strlen(insert);
274.            fark = patLength - insertLength;
275.
276.            printf("Lutfen acmak istediginiz dosyanin ismini uzantisiyla birlikte gir
    iniz.\n");
277.            scanf("%s", filename);
278.            if((fp = fopen(filename,"r")) == NULL) {
279.                printf("Dosya okunmak icin acilamadi!\n");
280.                return 0;
281.            }
282.            else {
283.                if(case_sensitive == 1) {   //arama case-
    insensitive mi kontrol eden if statement
284.
285.                    //aranacak kelimenin tum karakterleri kucuk karakter olacak sekil
    de kelimeyi yeniden duzenleyen dongu
286.                    for(i = 0; i < strlen(search); i++){
287.                        lowerSearch[i] = tolower(search[i]);
288.                    }
289.                    badMatch = createBadMatch(lowerSearch,bad_match);   //badmatch ta
    ble olusturulur
290.                    fseek(fp, 0, SEEK_END); // seek to end of file
291.                    size = ftell(fp); // get current file pointer
292.                    fseek(fp, 0, SEEK_SET); // seek back to beginning of file
293.                    fread(text,size,1,fp); //dosyadaki text text dizisine alinir.
294.                    printf("Text: %s\n",text);
295.
296.                    //dosyadan alinan text'in tum karakterleri kucuk karakter olacak
    sekilde text'i duzenleyen dongu
297.                    for(i = 0; i < strlen(text); i++){
298.                        lowerText[i] = tolower(text[i]);
299.                    }
300.                    //printf("Lower Text: %s\n",lowerText);
301.                    //printf("Lower pattern: %s\n",lowerSearch);
302.
303.                    start = clock(); //sure hesaplama baslangici
304.                    location = caseHorspool(lowerText,lowerSearch,badMatch,lokasyon);
     //pattern text'te aranir, count ve bulunan patternlerin yeri dondurulur.
305.                    if(location[0] == 0) {      //count 0 ise text'te aranan pattern
    mevcut degildir.
306.                        printf("Eslesme yok!\n");
307.                        return 0;
308.                    }
309.                    else {
310.                        j = 0;  //count-control
311.                        i = 1;  //location dizisinin ilgili gozune erisen indis numar
    asi.Boylece text'te hangi indisde degisim yapilacagi ogrenilir.
312.
313.                        //count kadar donerek swap islemi yaptiran while dongusu
314.                        while (j < location[0]) {
315.                            if(j == 0){
316.                                swap(location[i],fark,insertLength,text,search,insert
    );
317.                                j++;
318.                                i++;
```

```
                            }
                            else {
                                //patternlerin baslangic indislerini guncelleyen for dongusu
                                for(z = i; z <= location[0]; z++){
                                    location[z] = location[z] - fark;   //ilk degisimden sonra text dizisinde diger bulunan patternlerin yeri fark kadar kayacaktir. Bu islem sayesinde diger patternlerin yerleri dogru bir sekilde guncellenir.
                                }
                                swap(location[i],fark,insertLength,text,search,insert);
                                j++;
                                i++;
                            }
                        }
                        end = clock(); //sure hesaplama bitisi.
                        time_spent = (double) (end - start);
                        printf("Son Text: %s\nDegisim sayisi: %d\nGecen zaman: %f ms\n",text,location[0],time_spent);
                    }
                    fclose(fp); //dosyayi okumak icin acildigindan kapatir.

                    //dosyayi yazmak icin yeniden acar ve yeni text'i icine yazar.
                    if((fp = fopen(filename,"w")) == NULL) {
                        printf("Dosya yazmak icin acilamadi!\n");
                        return 0;
                    }
                    else {
                        fputs(text,fp);
                        fclose(fp);
                        printf("Dosya Icerigi Guncellendi!\n");
                    }
                }
                else {  //case sensitive durum.
                    badMatch = createBadMatch(search,bad_match); //bad-match table olusturulur
                    fseek(fp, 0, SEEK_END); // seek to end of file
                    size = ftell(fp); // get current file pointer
                    fseek(fp, 0, SEEK_SET); // seek back to beginning of file
                    fread(text,size,1,fp);
                    printf("Text: %s\n",text);
                    start = clock();    //sure hesaplama baslangici
                    count = horspool(text, search, insert, badMatch);   //text'te pattern bulunduysa ilgili degisiklik yapilir ve count degeri disari donulur.
                    if(count == 0){
                        printf("Eslesme yok!\n");
                        return 0;
                    }
                    end = clock();  //sure hesaplama bitisi
                    time_spent = (double) (end - start);
                    //printf("Count: %d\n",count);
                    printf("Son Text: %s\nDegisim sayisi: %d\nGecen zaman: %f ms\n",text,count,time_spent);
                    fclose(fp); //dosyayi okumak icin acildigindan kapatir.

                    //dosyayi yeniden acar ve yeni text'i icine yazar.
                    if((fp = fopen(filename,"w")) == NULL) {
                        printf("Dosya yazmak icin acilamadi!\n");
                        return 0;
                    }
                    else {
                        fputs(text,fp);
                        fclose(fp);
                        printf("Dosya Icerigi Guncellendi!\n");
                    }
```

```
376.                 }
377.             }
378.             return 0;
379.         }
```