

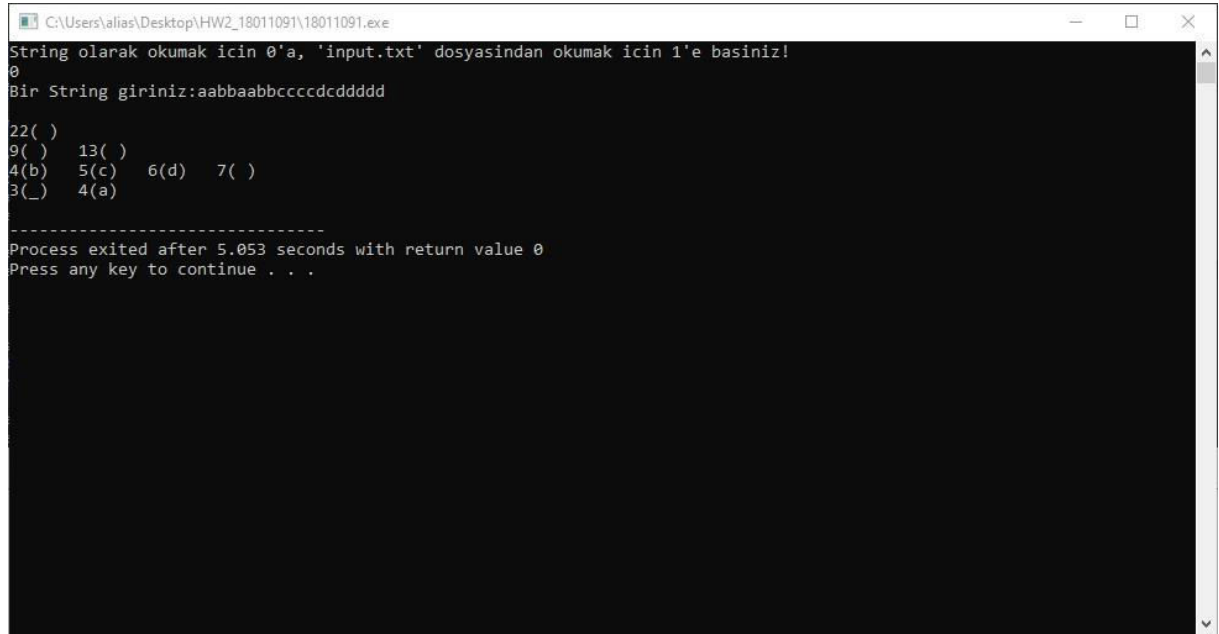
VERİ YAPILARI ÖDEV-2 RAPORU

Ekranda harfler ve frekansları, “**frekans(harf)**” şeklinde bastırılmaktadır. Eğer düğüm harfi belirtmiyorsa, yani çocuklara (child) sahipse “**frekans()**” şeklinde bastırılmaktadır.

Çıktıda boşluk karakteri daha kolay anlaşılabilmesi açısından “_” şeklinde kullanılmıştır.

Örnekler

1) aabb aabbcc cccd cddddd girdisi için çıktı:



```
C:\Users\alias\Desktop\HW2_18011091\18011091.exe
String olarak okumak için 0'a, 'input.txt' dosyasından okumak için 1'e basınız!
0
Bir String giriniz:aabbaabbcccdccddddd

22( )
9( )   13( )
4(b)   5(c)   6(d)   7( )
3(_)   4(a)

-----
Process exited after 5.053 seconds with return value 0
Press any key to continue . . .
```

Çıktıdan da görülebileceği üzere 4 adet a, 4 adet b, 5 adet c, 6 adet d ve 3 adet boşluk bulunmaktadır.

Verilen örnekte 22() düğümünün 2 tane çocuğu vardır ve bunlar alt satırda sırasıyla bastırılmıştır. 9() ve 13() düğümlerinin harfleri belirtmedikleri görülmektedir. Bunların da çocukları sırasıyla, ilk ikisi 9() düğümünün ve sonraki ikisi 13() düğümünün olmak üzere alt satırda bastırılmıştır.

3. satırda sadece 7() düğümünün çocukları olduğunu görmekteyiz. 7() düğümünün çocukları da alt satırda sırasıyla bastırılmıştır.

2) baddcccc girdisi için çıktı:

```
C:\Users\alias\Desktop\HW2_18011091\18011091.exe
String olarak okumak için 0'a, 'input.txt' dosyasından okumak için 1'e basınız!
0
Bir String giriniz:baddcccc

10( )
5(c) 5( )
2( ) 3(d)
1(a) 1(b)

-----
Process exited after 6.849 seconds with return value 0
Press any key to continue . . .
```

Çıktıdan da görülebileceği üzere 1 adet a, 1 adet b, 5 adet c ve 3 adet d bulunmaktadır.

Verilen örnekte 10() düğümünün 2 tane çocuğu vardır ve bunlar alt satırda sırasıyla bastırılmıştır. 5(c) ve 5() düğümlerinden 5() bir harfi belirtmemektedir. O nedenle 5()’nin çocukları vardır ve alt satırda sırasıyla bastırılmıştır.

3. satırda sadece 2() düğümünün çocukları olduğunu görmekteyiz. 2() düğümünün çocukları da alt satırda sırasıyla bastırılmıştır.

Fonksiyonlar

ekle fonksiyonu: Bağlı listeye yeni elemanları eklemektedir.

insertion_sort fonksiyonu: sorted_insert fonksiyonunu kullanarak bağlı listeyi sıralamaktadır.

sorted_insert fonksiyonu: Bağlı listeye sıralama amacıyla, elemanı araya sokma (insert) işlemi yapmaktadır.

huffman fonksiyonu: İstenildiği şekilde bir ağaç oluşturmaktadır.

bastir fonksiyonu: Oluşturulan ağacı ekrana bastırmaktadır.

İstenen Çıktı

“huffman coding is a data compression algorithm” girdisi için çıktı:

```
C:\Users\alias\Desktop\HW2_18011091\18011091.exe
String olarak okumak için 0'a, 'input.txt' dosyasından okumak için 1'e basınız!
1

46( )
18( ) 28( )
8( ) 10( ) 12( ) 16( )
4( ) 4( ) 5(a) 5( ) 6( ) 6( ) 8( ) 8( )
2(h) 2(n) 2(t) 2( ) 2( ) 3(m) 3(n) 3(s) 4(i) 4(o) 4( ) 4( )
1(e) 1(l) 1(p) 1(u) 2(c) 2(d) 2(f) 2(g)

-----
Process exited after 1.776 seconds with return value 0
Press any key to continue . . .
```

Yorum Satırları

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct n{
5      char harf;
6      int frequency;
7      struct n *left, *right, *next;
8  };
9  typedef struct n node;
10
11 void ekle (node** root, char harf, int frequency) {
12     node* new_node = (node *)malloc(sizeof(node)); //yeni düğüm oluşturuluyor
13     new_node->harf = harf; new_node->frequency = frequency; //harf ve harfin frekansı atanıyor
14     new_node->left = NULL; new_node->right = NULL; //kolları NULL olarak atanıyor
15
16     new_node->next = *(root); //yeni oluşturulan düğüm önceki düğüme bağlanıyor.
17     *(root) = new_node; //kök güncelleniyor
18 }
19 void sorted_insert(node**, node*);
20
21 void insertion_sort(node **root) {
22     node *sorted = NULL;
23     node *current = *root;
24
25     while (current != NULL) {
26         node *next = current->next; //iterasyon için next saklanıyor
27         sorted_insert(&sorted, current); //current sıralı linkli listeye gönderiliyor
28         current = next; //current güncelleniyor
29     }
30     *root = sorted; //root sıralanmış linkli listeyi göstermek üzere ayarlanıyor
31 }
32
```

```

33 void sorted_insert(node** root, node* new_node) {
34
35     node* current;
36     // yeni elemanın köke gelmesi durumu
37     if (*root == NULL || (*root->frequency >= new_node->frequency) {
38         new_node->next = *root;
39         *root = new_node;
40     }
41     //eklenecek yerin öncesi bulunuyor
42     else {
43         current = *root;
44         while (current->next!=NULL && current->next->frequency < new_node->frequency) {
45             current = current->next;
46         }
47         new_node->next = current->next;
48         current->next = new_node;
49     }
50 }
51
52 node* huffman (node *head_ref) {
53     node *iter, *yede;
54     int toplam;
55
56     if (head_ref->next == NULL){ //1. seviyede 1 düğüm olması durumu
57         return head_ref;
58     }
59
60     else if (head_ref->next->next == NULL){ //1. seviyede 2 düğüm olması durumu
61         toplam = 0;
62         node* new_node = (node *)malloc(sizeof(node));
63         toplam += head_ref->frequency;
64         toplam += head_ref->next->frequency;

```

```

65         new_node->frequency = toplam; //toplam değeri ilk 2 düğümden hesaplanıp yeni düğüme atanıyor
66         new_node->harf = ' ';
67         new_node->next = NULL; //1. seviyede 1 düğüm kalacağı için yeni oluşan düğüm NULL'ı gösteriyor
68         new_node->left = head_ref; //yeni düğümün soluna ve sağına değerler atanıyor
69         new_node->right = head_ref->next;
70         head_ref->next = NULL;
71
72         return huffman(new_node);
73     }
74
75     else{ //1. seviyede 2'den fazla düğüm olması durumu
76         int flag = 0, counter=0, i, toplam = 0;
77         node* new_node = (node *)malloc(sizeof(node));
78
79         //yeni düğüme atamalar yapılıyor
80         toplam += head_ref->frequency;
81         toplam += head_ref->next->frequency;
82         new_node->frequency = toplam;
83         new_node->harf = ' ';
84
85         iter = head_ref;
86
87         while (toplam >= iter->next->frequency && iter->next->next != NULL){
88             iter = iter->next; //iter, yeni eklenecek düğümün ekleneceği yeri gösteriyor
89             counter++; //counter değişkeni yeni eklenecek olan düğümün başa, ortaya veya sona geleceğini anlamak amacıyla kullanılıyor
90         }
91         while (iter->next != NULL && new_node->frequency >= iter->next->frequency){ //Üstteki while'da iter en son düğüme kadar gidemiyor
92             iter = iter->next; //en sona ekleme ayrı değerlendirildiği için eğer en sona düğüm eklenecekse bu while'a giriyor
93             counter++;
94             flag = 1; //flag değeri 1 olarak atanıyor
95         }

```

```

96         if (counter > 1 && iter->next != NULL){ //yeni eklenen düğümün başta veya sonda değil de ortada bir yerlerde ise,
97             flag = 2; //flag değeri 2 olarak atanıyor
98         }
99
100         new_node->next = iter->next;
101         new_node->left = head_ref;
102         new_node->right = head_ref->next;
103         if (flag == 1) { //yeni eklenen düğümün son düğüm olma durumu
104             node *head_ref2; //ilk iki düğüm toplanıp yeni bir ağaç elde ediliyor
105             head_ref2 = head_ref->next->next; //dolayısıyla yeni kök düğümü 3. sıradaki düğüm oluyor
106             head_ref->next->next = NULL; //yeni oluşan ağaçtan,
107             head_ref->next = NULL; //önceki ağaçtaki bağlantılar kaldırılıyor
108             head_ref = iter;
109             head_ref->next = new_node; //yeni düğümün bağlantısı yapılıyor
110             new_node->next = NULL;
111             return huffman(head_ref2); //yeni kök düğüm gönderiliyor
112         }
113         else if (flag == 2){ //yeni eklenen düğümün başta veya sonda değil de ortada bir yerlerde olma durumu
114             iter->next = new_node;
115             node *head_ref2 = head_ref->next->next;
116             head_ref->next->next = NULL; //yeni oluşan ağaçtan,
117             head_ref->next = NULL; //önceki ağaçtaki bağlantılar kaldırılıyor
118             return huffman(head_ref2);
119         }
120         else{ //yeni eklenen düğümün ilk düğüm olma durumu
121             head_ref->next->next = NULL; //yeni oluşan ağaçtan,
122             head_ref->next = NULL; //önceki ağaçtaki bağlantılar kaldırılıyor
123             head_ref = iter;
124             return huffman(new_node);
125         }
126     }
127 }
128

```

```

129 void bastir(node *root){
130     int front=0, fr=0, i;
131     node *queue[1000];
132     queue[front] = root;
133     front = front+1;
134     queue[front] = NULL;
135
136     while(fr <= front) {
137         node *temp=queue[fr];
138
139         if(queue[fr] == NULL && queue[fr-1] == NULL) {
140             break;
141         }
142         fr = fr+1;
143         if(temp == NULL) { //temp değişkeni NULL'a eşitse,
144             front = front+1; //o seviyede başka düğüm kalmadı demektir
145             queue[front] = NULL;
146             printf("\n"); //ve bir alt satıra geçilir.
147         }
148         else{
149             printf("%d(%c) ",temp->frequency, temp->harf); //düğüm eğer varsa harfiyle beraber basılıyor
150             if(temp->left) { //düğümün solunda eleman varsa,
151                 front = front+1;
152                 queue[front] = temp->left; //o düğüm kuyruğa ekleniyor
153             }
154             if(temp->right) { //düğümün solunda eleman varsa,
155                 front = front+1;
156                 queue[front] = temp->right; //o düğüm kuyruğa ekleniyor
157             }
158         }
159     }
160 }
161

```

```

162 int main () {
163     node *root = NULL, *root2, *root3=NULL;
164     char string[150], ch;
165     int n=0, i, h, secenek=3, c = 0, frequency[27] = {0};
166     FILE *fp;
167
168     while (secenek != 0 && secenek != 1){
169         printf("String olarak okumak için 0'a, 'input.txt' dosyasından okumak için 1'e basınız!\n");
170         scanf("%d",&secenek);
171         if (secenek != 0 && secenek != 1)
172             printf("\nHatalı bir giriş yaptınız! Lütfen tekrar deneyiniz!\n");
173     }
174
175     if (secenek == 0){
176         ch = getchar();
177         printf("Bir String giriniz:");
178         gets(string);
179     }
180
181     else{
182         fp = fopen("input.txt","r"); //Dosya sorunsuz bir şekilde açılmadıysa program devam edemez
183         if (fp == NULL){
184             printf("Dosya açilamadil!");
185             return 0;
186         }
187         else{
188             do {
189                 fgets(string,150,fp);
190             } while (!feof(fp));
191         }
192         fclose (fp);
193     }
194

```

```

195     while (string[c] != '\0') { //String'in sonuna gelene kadar işlem devam ediyor
196         if ((string[c] >= 'a' && string[c] <= 'z') ) //burada string'deki harflerin frekansı hesaplanıyor
197             frequency[string[c] - 'a']++;
198         else if (string[c] == ' ') //burada da girdideki boşluk karakterlerinin frekansı hesaplanıyor
199             frequency[26]++;
200         c++;
201     }
202     for (i = 0; i < 26; i++) { // harfler frekanslarıyla birlikte linkli listelere gönderiliyor
203         if (frequency[i] != 0)
204             ekle(&root, i+'a', frequency[i]);
205     }
206     if (frequency[26] != 0) // boşluk karakteri ayrıca değerlendiriliyor
207         ekle(&root, '_',frequency[26]);
208
209     printf("\n");
210     insertion_sort(&root); // listeler insertion sort ile frekanslarına göre sıralanıyor
211     root = huffman(root); //ağac istenen şekile getiriliyor
212     bastir(root); //ağaç bastırılıyor
213
214     return 0;
215 }

```