

# T.C. SAKARYA ÜNİVERSİTESİ

# BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

## VERİ YAPILARI ÖDEV RAPORU

NUMARA:.....B191210008

AD:.....Gamze

SOYAD:....Ceylan

**GRUP:.....1/B** 

#### Özet

Sayilar.txt dosyasındaki # karakteri ile ayrılmış olan sayılar sağ ve sol kısım ayrı olcak şekilde okunacak, her 3 basamaklı sayı listeye eklenecek ve düğüm verileri karşılıklı karşılaştırılacak. Bu karşılaştırma sonucuna göre de çeşitli işlemler yapılacak.

#### 1. GELİŞTİRİLEN YAZILIM

Öncelikle Node ve DoubleLinkledList sınfını oluşturdum. Bir liste oluşturmak ve ödevde istenen işlemleri yapmak için gerekli metotları ekledim.

Ayrıca dosyadan okunan verileri üç basamaklı sayıya çeviren bir Number sınıfı oluşturdum

```
// kurucu fonksion
// dosyadan char olarak aldığı elemanları üç basamaklı sayıya dönüştürüyor
Number::Number(char units, char tens, char hundreds) {
    this->units = units - 48;
    this->tens = tens - 48;
    this->hundreds = hundreds - 48;
    if(this->hundreds == 0) this->hundreds = 1;
}

// üç basamaklı sayıyı döndürüyor
    int Number::getNumber() const {
        return hundreds*100 + tens*10 + units;
    }
```

Ödevde özellikle istenen reverse() ve swap() metotlarını yaparken aşağıdaki şekilde bir yol izledim:

#### a) swap()

```
// gönderilen indexe göre iki listenin karşılıklı elemanlarını değiştiriyor
    void DoubleLinkedList :: swap(int index, DoubleLinkedList*& list1, DoubleLinkedList*& list2){
   Node *temp1 = list1->FindByPosition(index);
        Node *temp2 = list2->FindByPosition(index);
        if(index == 0){
            Node *head11 = list1->FindByPosition(index+1);
            Node *head22 = list2->FindByPosition(index+1);
             temp1->next = head22;
            head22->prev = temp1;
             temp2->next = head11:
            head11->prev = temp2;
             list1->head =temp2;
            list2->head = temp1;
        else if (index == list1->Count()-1) {
   Node *head1 = list1->FindByPosition(index-1);
            Node *head2 = list2->FindByPosition(index-1);
            head1->next = temp2;
             temp2->prev = head1;
            head2->next = temp1;
             temp1->prev = head2;
             Node *head11 = list1->FindByPosition(index+1);
            Node *head22 = list2->FindByPosition(index+1);
             Node *head1 = list1->FindByPosition(index-1);
             Node *head2 = list2->FindByPosition(index-1);
            head1->next = temp2;
temp2->prev = head1;
             temp2->next = head11;
            head11->prev= temp2;
            head2->next = temp1;
             temp1->prev = head2;
             temp1->next = head22;
             head22->prev= temp1;
```

Önce değiştirilmesi istenen indexteki düğümleri tuttum.

Daha sonra gönderilen index'in liste başı, liste ortası veya liste sonu olabileceğini düşünerek fonksiyonu tasarladım.

Düğümler değiştirilirken verinin kaybolmaması için ilgili düğümün bir önceki ve sonraki adresini tuttum. Son olarak karşılıklı düğümleri değiştirdim ve listeleri bağladım

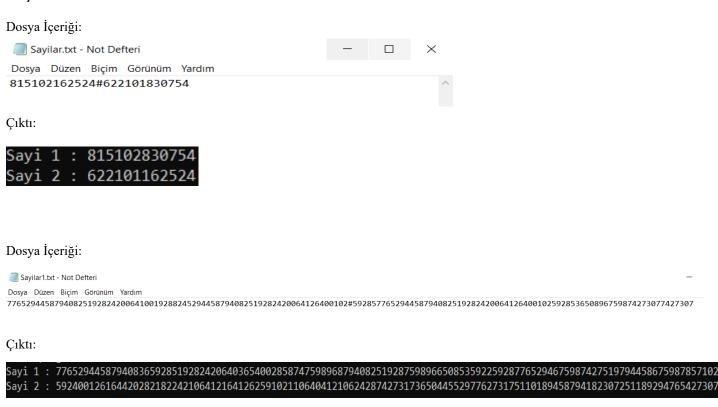
#### b) reverse()

```
// listeyi ters çeviren fonksiyon
void DoubleLinkedList :: reverse() {
    Node *itr = head;
    Node *temp;
    Node *tmpL = FindByPosition(size);
    for(int i=0; i<size; i++) {
        temp = itr->next;
        itr->next = itr->prev;
        itr->prev = temp;
        itr = temp;
    }
    head=tmpL;
}
```

Liste başını kaybetmemek için itr'i, bir sonraki düğümü kaybetmemek için temp'i ve liste liste sonunu head olarak güncellemek için de tempL yani liste sonunu tuttum.

Daha sonra düğümlerin next ve prev'lerini değiştirerek çevrime işlemini yaptım Son olarak da liste başını güncelledim.

#### 2. ÇIKTILAR



### 3. SONUÇ

- Verilen ödev ile birlikte bağıl liste veri yapısını oluşturmayı öğrendim ve mantığını kavradım.
- Fonksiyonları yazarken arka planda işlemlerin nasıl yapıldığını gördüğüm için hangi durumlarda bağıl liste kullanırsam daha avantajlı olacağını anladım.
- Heap bellek bölgesindeki işlemlerin nasıl gerçekleştiğini daha iyi anladım.