**Sınıf Planlama Sistemi**

[**Nedret Gegeoğlu**](mailto:nedretgegeoglu@windowslive.com)

[**Selda Yapal**](mailto:seldayapal@anadolu.edu.tr)

**Kocaeli Üniversitesi**

**Bilgisayar Mühendisliği 2017 – 2018**

**Proje Özeti:**

Projede temel olarak bizden istenen; txt dosyasında verilen düzensiz öğrenci listesini düzenleyip iki farklı dağılım ile sınıflara ayırmaktı. Proje C dili üzerinde gerçeklendi ve bunu yaparken herhangi bir ekstra kütüphaneye ihtiyaç duymadık.

Projede öğrenmemiz gereken dosya işlemleri üzerinde çalışıldı. Bunun haricinde **C** dili nesneye yönelik bir dil olmamasına rağmen buna en yakın kullanılabilir metot **struct** metoduydu ve struct üzerinde öğrenci listesini aktarıp, oluşturup, düzenledik.

Sonuç olarak karışık ve düzensiz bir öğrenci listesini gözden geçirip, düzenleyip, sıralayan ve aktaran dinamik bir projeye sahibiz.

**1.Giriş:**

İlk olarak bizden istenen txt dosyasındaki verileri içe aktarmaktı. Bunun için **FILE** işlemleri ve **String** metotları hakkında bilgi sahibi olmalıyız. Bu konular hakkında bilgi sahibi olduktan sonra nihayet dosyayı içeri aktarabildik.

Daha sonra **Struct** metodu sayesinde her öğrencinin bilgilerini ayrı ayrı kaydettik ve bu struct üzerinde işlemler yapmaya başladık:

Bu işlemler öğrencilerin hatalı bilgilerini düzeltme, eksik bilgilerini doldurma, belirli kıstaslara göre sıralama yapmak üstüneydi.

Son olarak kullanıcıdan istenen bilgiler karşılığında öğrenciler sıralı bir şekilde iki türde farklı sınıflara aktarıldı. Bunlardan:

1.Eşit Dağılım yöntemi:

Öğrencileri girilen sınıf adedi ve sınıf mevcutlarına göre ve öğretim türüne ayırarak ortalama mevcut bulup yerleştirme yöntemi.

2.En az sınıf dağılım yöntemi:

Öğrencileri(öğretim türüne göre) girilen sınıf ve mevcut durumuna göre en yüksek mevcutlu sınıftan atamaya başlayarak en fazla boş sınıf bırakmaya yönelik bir yöntemdir.

**2.Temel Bilgiler:**

Projeyi **C** dili üzerinde **Codeblocks** **IDE**’sinde derledik.

* C dili içerisinde txt işlemleri için kütüphane içerisinde **FILE** fonksiyonları kullanıldı.
* Nesne kavramına uygun olması için **Struct** metodu kullanıldı.

**3.Diğer Bölümler:**

**Projede Kullanılan Fonksiyonlar:**

3.1. SatırNoBul():

Verilen **ogrenci\_kayit\_bilgileri.txt** dosyasında alt satıra geçerek ve geçtiğinde satır numarasını artırarak bulmasını sağlayan bir fonksiyon oluşturuldu. Amacı; dosyanın temel formatına uygun olarak txt dosyası içerisinde kaç adet kayıtlı öğrenci olduğunu bulmaktı.

3.2. Atama(struct ogrenci \*ogr):

Bu fonksiyonda bir önceki satirNoBul(); fonksiyonundan elde ettiğimiz satır numarasını alarak, ilk satırdaki bilgi girdilerini atlayarak her satırdaki öğrenciyi oluşturulan **struct ogrenci** nesnesinin girdilerine aktardık.

Struct Ogrenci içerisindekiler:

* Int ogrNo; Öğrenci Numarası
* Int k\_sirasi; Öğrenci kayıt sırası
* Char Ad; Öğrenci isim
* Char Soyad; Öğrenci soy isim
* Char ogretim; Öğrenci öğretim türü

Bu fonksiyonda ayrıca yazılım içerisinde kullanıcıya yeni kayıt yapma imkanı da sunduk. Bu metodun çalışma prensibi; kullanıcı **while** döngüsü içerisinde yeni kayıt yapmak istediği sürece yeni kayıt yapabilir şeklinde düzenlendi. Ayrıca yapılan yeni kayıtların uyması gereken başlıca kıstaslarda mevcut. Bunlar:

* Öğrenci numarası: Kullanıcı girdiği öğrencinin öğrenci numarasını bilmediği takdirde öğrenci no kısmını ‘0’ girerek geçer ve yazılım öğrenci numarasını girilen kayıt sırası ve öğretim türüne göre düzenler. Öğrenci numarası girilmiş ise kullanıcıya kayıt sırası ve öğretim türü sorulmaz. Listede mevcut olan bir öğrenci numarası giremez, tekrar numara girmesi gerekir.
* Öğrenci adı ve soyadı benzerliği kıstasına dikkat edilmemiştir bunun sebebi böyle bir durumun mümkün olabilmesi yüzündendir.
* Kayıt Sırası: Öğrencinin öğrenci numarası yoksa kayıt sırası girmesi mecburidir. Öğrenci numarası varsa bu kısım es geçilir.
* Öğretim Türü: Öğrencinin öğrenci numarası yoksa öğretim türü ( I/II) şeklinde girmesi istenir. Öğrenci numarası varsa bu kısım es geçilir. Kullanıcının muzipliğine göre isterse öğrencinin IV. Öğretim olduğunu da girebilir. Böyle bir kısıtlama getirmedik. Yazılım onu da listelemeye dahil eder.

Bu kriterler gerçekleştikten sonra yeni öğrenci kaydetmek için kullandığımız indis değişkeni artar ve bu değişkeni; txt dosyasından elde edilen satır numarasına ekleriz. Bu işlemi gerçekleştirmek için oluşturduğumuz struct ogrenci ogr[1000] nesnesinde [900] numaralı indisin öğrenci numarasına aktarıp diğer fonksiyonlara böyle taşıdık. **Atama();** fonksiyonunu **void** fonksiyon türünden **int** fonksiyon türüne değiştirdiğimizde birden fazla fonksiyonun return değerini tek bir fonksiyonda çektiğimiz için bazı hatalarla karşılaştık ve bu yüzden böyle bir taşıma(çağırma) yöntemi bulduk.

3.3. TaraVeDuzenle(struct ogrenci \*ogr);

Bu fonksiyonda ilk olarak listede birden fazla kez adı geçen kayıtları listeler. Bunu kontrol etme şekli if koşullarıyla gerçekleştirildi.

* **Eğer**: iki öğrenci numarası birbirine eşitse ve öğrenci numarası sıfırsa bu öğrencinin ad ve soyadı aynı ise aynı öğrenciyi kaydettiğini anlıyoruz. Hata kodu olarak: “-(Hata ismi: isim soy isim benzerliği) … öğrenci numaralı öğrenci birden fazla kez kaydedilmiş.” Sonucunu verir. Yakalanan ikinci öğrenci değerleri silinip boş bırakılan yere bir alt sıradaki öğrenci yerleştirilir ve tüm öğrencileri kontrol etmek için satır numarası üzerinden işlem yapan döngüyü yakalanan öğrenci kadar az döndürmek için yakalanan değişkenini bir(++) artırırız.
* **Eğer**: İki öğrenci numarası birbirine eşitse ve bu numaralar gerçekten formata uygun öğrenci numaraları (‘0’ olmayan) ise Hata kodu olarak: “(hata ismi: eşit öğrenci numarası) … öğrenci numaralı öğrenci birden fazla kez kaydedilmiştir.” sonucunu verir. Tekrar yakalanan değişkeni bir artar ve kayıtlı tekrar eden ikinci öğrenci yerine bir alt satırdaki diğer öğrenci yerleştirilir.

Tekrar eden öğrenciler silindikten sonra liste güncellenir:

Burada ilk olarak yapılacak işlem öğrenci numaraları ‘0’ olan öğrencilere yeni öğrenci numaralarını vermektir. Burada koşullar girilen öğrenci numarasını örneğin; 1702001 numarası 3 koşula ayırırsak:

* 17: Giriş yılı
* 02: Öğretim türü
* 001: Kayıt sırası

Olur. Burada txt dosyasından aktarılan verilerden yararlanarak öğrencinin kayıt sırası ve öğretim türündeki girdiler numara oluşturulmasını sağlar. Girilen yeni öğrencinin ve numarası olmayan kayıtlı öğrencilerin bulunduğumuz sene (2017) içinde kayıt olduğunu düşündüğümüz için; kullanıcıya “kayıt yılı girin” gibi bir seçenek sunmadık.

Burada kayıt sırasını numaraya aktarmak için girilen numarayı %1000( 1000’e modlama) işlemi yaptık.

Daha sonra numaralı verilmiş listeyi komut ekranına yazdırdık.

3.4. FonkDevam(struct ogrenci \*ogr);

Bu fonksiyon temel olarak taraVeDuzenle(); fonksiyonunun devamı olarak oluşturuldu. Sebebi ise çok fazla işlem içerdiği ve karışık hale geldiği için ayrı bir görünüm vermekti. Burada taraVeDuzenle(); fonksiyonundan kullanmamız gereken tek değişken **yakalanan** değişkeniydi ve onuda **int** fonksiyon özelliği olan **return** ile aldık.

Bu fonksiyonda yaptığımız işlemler:

1.Öğrenci numaralarını küçükten büyüğe sıralandırma işlemi:

* İlk olarak en küçük eleman bulundu ve oluşturulan sıralama dizisinde ilk sıraya indis numarası yerleştirildi.
* En küçükten büyük, en büyükten küçük aralığında **for** döngüsü yardımı ile en küçükten en az büyük olan elemanlar sıra ile listelendi.
* Listenin güncel hali ekrana yazdırıldı.

2.Öğrencileri öğretim türüne ayırma işlemi:

* If koşulu yardımıyla öğrencinin öğretim türünü I veya II olduğunu bulup bu öğrencilerin struct dizisindeki indislerini ogr\_bir & ogr\_iki dizilerine ogretim\_bir & ogretim\_iki indis değişkenleri yardımıyla aktardık.
* Toplam kayıtlı olan öğrenci sayısını yazdırıp 1. Ve 2. Öğretimde kaç öğrenci mevcut ekrana yazdırdık.

3.Öğrencileri Öğretim türlerine göre girilen sınıf ve mevcutlara ayırma işlemleri:

* Kullanıcıdan istediği sınıf sayısı ve mevcutları alındı.
* Bu sınıfların adedine göre 1. Ve 2. Öğretim öğrenci sayısının ortalaması çıkarıldı.(Eşit Dağılım için)
* Girilen mevcut sınıf adedinin her iki öğretime de txt dosyaları oluşturuldu. Örneğin; **sinif1\_ogr2.txt** gibi.
* Eşit dağılım sıralaması için ortalamanın yanında kalan öğrenciler için kalan değişkeni oluşturulup işleme sokuldu. Örneğin:

17 öğrenci var ve 3 sınıf istendi. Ortalama; 5 olurken kalan; 2dir. Bu değişken yardımı ile bu 2 kişiyi 2 sınıfa birer birer aktardık. Ve eşit dağılım 6 6 5 şeklinde gerçeklendi 7 5 5 şeklinde değil.

* Öğretim bir ve ikiyi küçükten büyüğe indis numaralarının içerdiği diziler oluşturduk ve her iki öğretimin sıralamasına sahip olduk.
* **Toplam mevcut** değişkeni ile kullanıcıdan alınan sınıf mevcutları toplanıp öğrencilere yetecek mevcut var mı kontrol edildi. Yok ise kullanıcıdan tekrar mevcut istendi.
* Küçük mevcutlu sınıflar oluşturuldu (1. Ve 2. Öğr. İçin). Burada amaç eşit dağılım için kullanıcının girdiği sınıflardan ortalamanın altında kalan sınıfları listelemektir. Eğer “eşit dağılım ortalaması + kalan öğrenci değişkeninden gelen değer > sınıf mevcudu” bu diziye atanır.
* Kullanıcıdan dağılım tercihi alınır Eşit Dağılım için: 1 / En az sınıf dağılımı için: 2 buna göre dağılım işlemi başlatılır.

**3.4.1.Eşit Dağılım:**

**Eğer** girilen tüm sınıflar eşit dağılım ortalamasından yüksekse ( küçük mevcutlu sınıflar(kms)= 0):

* 1. Öğretim ve 2. Öğretime ortalamadan büyük sınıflara sahip oldukları için direkt olarak atama yapılır. Örneğin: Kullanıcı 3 sınıf girdi ve her sınıf mevcudunu 10 girdi ve bizim ortalamamız 1. Öğretim için beş 2. Öğretim için 4 ise. 1. Öğretim ataması: 5 5 5 2. Öğretim ataması: 4 4 4 şeklinde olur. Yada öğrenci ortalaması bölüm halinde kalan üretiyorsa örneğin; 1. Öğretimde 17 öğrencimiz varsa, kalan değişkeni sınıfa atama yapıldıktan bir eksilir ve eşit dağılım düzenini bozmaz. 1. Öğretim ataması: 6 6 5 şeklinde olur.
* Ekrana “Öğrenciler başarıyla atanmıştır” uyarısı gelir.

**Eğer** girilen sınıflardan eşit dağılımın ortalamasının altında kalan sınıflar (küçük mevcutlu sınıflar(kms) != 0) varsa:

* 1. Ve 2. Öğretime **ortalamadan büyük sınıflar** dizisi ve **atama yapılan sınıflar** dizisi oluşturulur.
* **For** döngüsü içerisinde girilen sınıf sayısı kadar döner. Her dönüşte tekrar eşit dağılım için ortalama alır. Bunun sebebi: Ortalaması büyük sınıf bulunup, daha önce atama yapılmadığı tespit edilip, atama yapıldıktan sonra azalan öğrenci sayısı ve azalan sınıf mevcudu sayısı yüzündendir. Burada gereken kıstasları sağlarsa sınıfa atama yapılır.
* Aynı işlemler 2. Öğretim içinde gerçeklenir.
* Uyarı (tüm öğrenciler başarı ile aktarıldı.) mesajı verir.

**3.4.2. En Az Sınıf Dağılımı:**

**Eğer** Dağılım tercihi olarak bu işlem seçilmişse:

* Girilen sınıflar büyükten küçüğe sıralanır. En büyük eleman bulunur. Küçükten büyük en büyükten küçük en büyük sınıflar sıralanır.
* En büyük sınıftan başlayarak atama işlemi mevcuda göre başlar, atanan öğrenci sayısı güncellenir ve öğretim türüne göre tüm öğrenciler atandığında durur.
* 2. Öğretim içinde aynı işlemler yapılır.
* Uyarı: “atama yapılan sınıf … atama yapılan öğrenci sayısı …” şeklinde ekrana yazdırılır.

**4.Sonuçlar:**

Proje genel olarak bilgi konusunda basit ama isterler konusunda karışık bir projeydi. Genel olarak hemen hemen her şeyi gerçekleştirdik. Sıralama düzenlerinin bazı şekillerinde sorunlarla karşılaşıp çözüp, çözemediklerimiz oldu. Yinede **dosyalama**, **string** ve **struct** konularında gelişim için uygun bir proje olduğunuz düşünüyoruz.

**5.Kaynakça:**

[1]. Stringler - Embedded Koü : [https://docs.google.com/document/d/1OH\_Pa2Qr7wphbrpm63kxrqj2NjoxtobCWdKDlrJnLLU/edit](https://docs.google.com/document/d/1OH_Pa2Qr7wphbrpm63kxrqj2NjoxtobCWdKDlrJnLLU/edit%20)

[2]. Struct – Embedded Koü :

<https://docs.google.com/document/d/1wEdOU1DHULp7nQQc7Xa2w1QGzJWnRR5FnNj6Z4E8E8Y/edit>

[3]. Dosyalama – Embedded Koü :

<https://docs.google.com/document/d/1SvmffN6pwUwu76FaWaY1zCFeAwWdXiSsk2PeDitcL8M/edit>