Veri yapıları Ödev 1 Raporu

Yazarlar: İsmet GÜZELGÜN(b121210025@sakarya.edu.tr)
İbrahim AKDAĞ(b121210027@sakarya.edu.tr)

Özet

Ödevde C++ programlama diliyle, pointerlar yardımıyla nesneye dayalı programlama paradigmasına uygun bir şekilde üç adet sınıf oluşturmamız istendi. Programda üç adet çivi ve her çiviye asılı bir sepet oluşturulması ve kullanıcıdan alınan veriler doğrultusunda çivilerin yerleri değiştirilmemek koşuluyla sepetlerin yerlerinin SepetKontrol.cpp sınıfı yardımıyla değiştirilmesi isteniyordu. Ödevde istenilenleri yerine getirdiğimizi düşünüyoruz. Açıklamalar aşağıdadır.

Sepet.cpp

Sepet sınıfında nesneye dayalı programlama paradigmasını hakim kılmaya çalıştık. Örneğin SepetCiz() fonksiyonu sadece bizden ekrana çıkartılması istenilen şekli çizdirmekle sorumluydu.

```
 */
void Sepet::SepetCiz()
{
    cout<<  "  .|.  "<<endl<<" .   . "<<endl<<".     ."<<endl<<"#     #"<<endl<<"#  "<<SepetAdiBelirle(sepNo)<<"  #"<<endl<<"#     #"<<endl<<"#######\n";
}
```

Yine üç farklı sepet çizdirip A,B,C değerlerini bu üç farklı sepete çizdirmek düşüncesi bahsedilen paradigmaya aykırı olacağı için hangi harfin hangi sepete getirileceğini belirlemek için bir SepetAdiBelirle(int) fonksiyonu yazmamız gerekti. İçerisinde basit bir switch mekanizması olan bu fonksiyon ödevin verildiği ilk gün başta sadece ekrana sepetleri çıkartabilmek için kullanıldı. Fakat sonrasında ödevin bitirilmesi için gerekli olduğuna karar verip kullanmaya devam ettik.

```
char Sepet::SepetAdiBelirle(int n)
{
    switch(n)
    {
        case 0:
        {
            sepAd='A';
            return sepAd;
        }
        case 1:
        {
            sepAd='B';
            return sepAd;
        }
        case 2:
        {
            sepAd='C';
            return sepAd;
        }
    }
}
```

Sepet(int) kurucu fonksiyonu integer türünden bir parametre almakta ve buradan gelen değeri private olarak tanımlanmış sepNo yani sepet numarası değişkenine atamaktadır. Bunun sebebi SepetKontrol.cpp sınıfında oluşturulan her bir çivi nesnesi için döndürülen index değeri ile çivi nesnelerinin kurucularında yer alan sepet nesnelerinin ilerleyen aşamalarda kullanılabilmesini kolaylaştırmaktır.

Civi.cpp

Probleme nasıl yaklaşmamız gerektiğini anladıktan sonra Civi.cpp sınıfını oluşturmak bizim için pek problem olmadı. Yukarıda bahsettiğim üzere bu sınıf private olarak bir adet Sepet cinsi pointer ve bir integer türünden civiNo isimli değişkeni tutmakta ve sepet sınıfını manipüle edebilmek ya da SepetKontrol.cpp sınıfına yardımcı olabilmek adına gereken fonksiyonları içermektedir. Burada daha sonra karşılaştığımız çok önemli bir sorunu çözmek üzere kullanacağımız CiviyeSepetAdiDondur() fonksiyonundan bahsetmemiz gerekiyor. Bu fonksiyon sınıfın oluşturduğu sepet nesnesinden nesnenin sahip olduğu harfi döndürmesini talep eden basit bir fonksiyon olmakla beraber varlığı yok

sayıldığı ya da eklenmediği durumda SepetKontrol.cpp sınıfı için sıkıntı yaratıcı birkaç sorun oluşmasına neden oluyor. Bu konuya SepetKontrol.cpp sınıfını anlatırken ayrıca değineceğiz.

```
char Civi::CiviyeSepetAdiDondur()
{
    sepet->SepetAdiDondur();
}
```

SepetKontrol.cpp

En çok zamanımızı bu sınıfın tasarımı aldı diyebiliriz. Zira bir takım tasarım sorunları burada baş gösterdi. Yazdığımız program kullanıcıdan char olarak aldığı A,B,C harflerini önce integer türünden bir değişkene çeviriyor ve sonrasında bu integer değişkenleri bir pointer dizisi olarak oluşturduğumuz Civi* civiler[3] dizisinde kullanabiliyordu. Bu sırada sepet sınıfının içerisinde tekrar bu integer değişken char türüne çevrilerek gerekli değişim yapıldıktan sonra ekrana yine sepet adı olarak harflerin basılabilmesini sağlıyordu. Bu aşamada SepetKontrol.cpp sınıfında gerekli değişim işleminin yapılması gerektiğini kavramıştık fakat yaptığımız tasarım bizi oldukça zor durumda bırakmıştı. En başta ekrana sepetleri yazdırırken bir sorun olmuyordu. SepetKontrol.cpp sınıfının Degistir(int,int) fonksiyonunu çağırdığımızda yine bir sorun yoktu sepetler yer değiştiriyor adreslerin yeri değişiyordu. Fakat iş harflerinde adreslerle beraber hareket etmesine geldiğinde tasarımımız apaçık hatalıydı. Önce sorunun bununla sınırlı olduğunu düşündük fakat durum daha vahimdi. Zira kullanıcıdan tekrar değer aldırıp fonksiyonları değiştirmek istediğimizde A,B,C değerleri için sırasıyla 0,1,2 değerlerini yeniden atıyor sepet numaralarını kullanmak yerine en başta çivilerle aynı index değeriyle atattığımız sepetleri tekrar aynı sırayla yani 0,1,2 şeklinde sıralayıp ilk sırada(yani 0) C ya da B sepeti olsa bile onu A sepeti, keza ikinci sırada(yani 1) A ya da C sepeti olsa bile bunu B sepeti olarak algılayıp buna göre işlem yapıyordu. Sorunu başta CiviNumarasi(char) şeklinde yazdığımız bir SepetKontrol.cpp fonksiyonu yaratıyordu.

```
int SepetKontrol::CiviNumarasi(char ci)
{
    switch(ci)
    {
        case 'A': return 0;
        case 'B': return 1;
        case 'C': return 2;
    }
}
```

Bu fonksiyon Test.cpp içerisinde her çalıştığında kullanıcı A girdiği durumlarda 0,B girdiğinde 1 ve C girdiğinde 2 atıyordu. Bu da çivilerin ve dolayısıyla sepetlerin yanlış sıralanmasına neden oluyordu. Bu sorunu çözmek yaklaşık iki gecemizi aldı. En sonunda yine SepetKontrol.cpp sınıfı içerisinde bir fonksiyon yazarak alınan her kullanıcı değerini mevcut sepet isimleriyle karşılaştırarak bu işi çözdük.

```
int SepetKontrol::DogruIndexiDondur(char dogruMu)
{
    for(int i=0;i<3;i++)
    {
        if(civiler[i]->CiviyeSepetAdiDondur()==dogruMu)
            return i;
    }
}
```

Sonuç

Geçen yıl bu derste oldukça zorlanmış bizler için bu ödev yine zorlayıcı oldu. Özellikle yaptığımız tasarımın işini gerektiği gibi gerçekleştiremediğini gördüğümüzde problemin nerede olduğunu anlayabilmekte zorlandık. Netice itibariyle bizim görebildiğimiz kadarıyla tasarımımızın son halinin bir problemi olmamakla beraber eksik bir yer bırakılmadığına inanıyoruz.