T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR ve BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

VERİ YAPILARI DERSİ

FİNAL ÖDEVİ – RAPOR

AVL AĞACI İLE DÜĞÜM VE YIĞIT KULLANARAK POSTORDER SIRALAMA YAPAN C++ KONSOL UYGULAMASI

Hazırlayan Öğrencinin;

Adı: Oğuzhan

Soyadı: Tohumcu

Numarası: B181210397

Serdivan / SAKARYA

Ağustos, 2020

İSTENENLER ve ALGORİTMA

Ödevde AVL ağacının farklı bir uygulanışını tasarlıyoruz. AVL ağacının düğümleri bu sefer sadece int değerini tutmak yerine, bir kişi objesi ve hareketleri takip eden bir yığın objesini tutuyor. Ödevi yaparken uygulanışı kolay olması açısından ihtiyaç duyulan her şeyi sınıf tanımlayarak yapmaya çalıştım.

Öncelikle bir **Person** sınıfı oluşturdum bu sınıfta kişinin adı (*name*), kilosu (*weight*), yaşı (*age*) ve doğum tarihi (*dateOfBirth*) yer alıyor. *Person* sınıfı oluşurken isim (*name*), doğum tarihi (*dateOfBirth*) ve kilo (*weight*) gibi gerekli bütün bilgileri, yapıcı metot için vermek gerekiyor ve *Person* objesi oluşturulurken kişinin yaşı otomatik olarak hesaplanıyor. Ekleme işleminin nereye uygulanacağını saptamak için bu yaş parametresi kullanılıyor. Ekleme operasyonu gerçekleşirken; denge işlemine göre de bulunduğu düğümün yığın yapısına, düğümün hareketini ifade eden karakter atanır.

AVLTree kaynak dosyasındaki AVL ağacının yapısını tanımlayacak olursak; içerisinde Person objesi, Stack objesi, sağ ve sol düğümleri tutmak için düğüm işaretleyicileri, yükseklik, level ve güncellenmeden önceki level'i tutan yapılar vardır. Program süresinde çöp oluşmaması için genellikle objeleri işaretleyicilerle oluşturdum. Ağaca ekleme kişinin yaş değerine göre gerçekleşiyor, standart AVL ekleme işlemi yapılıyor. Sağ ve sol dengelerinin değişimlerine göre de stack objesine karakterler gönderiliyor.

ProgramController kaynak dosyasında tutucu ile bir AVL ağaç objesi üretiyorum. Yine ProgramController'da "Kişiler" dosyasını okuyup her sıra için bir Person objesi oluşturuyorum ve her sıra sonunda ağaca ekleme işlemi yapıyorum. Kişiler dosyasından gerekli bilgileri parçalamak için getline() fonksiyonunu kullanıyorum. İnternette yaptığım araştırmalar sonucu bu yöntemi keşfettim. Dosyadan alınan satırı "#" işaretine göre ayırma işlemini uyguluyorum.

Ağaç düğümlerinde yapılan değişiklikler ekleme sırasında düğümün yığın (heap) objesinde tutulur. Düğüm seviye olarak aşağı inerse, yığıta A harfi yüklenir. Seviye olarak yukarı çıkarsa, Y harfi yığıta atılır. Her bir ekleme işlemi bittikten sonra düğümlerin seviyeleri tekrar hesaplanır ve değişmeyen düğümlere de "D" karakteri atanır. Çift çevrimlerde ise, yığıta atılacak hareket sayısı bir olmuştur. Yani bir düğüm iki seviye yukarı çıksa bile yığıta sadece bir Y harfi atanmıştır.

YAPILANLAR, ZORLANILAN KISIMLAR ve SONUÇ

Projeyi yaparken 5 dosyaya bölmek istedim. Program (*Main.cpp*) dosyası ve 4 adet header dosyası. Yığıt yapısının bulunduğu (*Stack.hpp*) dosyasında; Stack'e ait constructor, destructor ile push, pop, ekrana yazdırma ve boş olup olmadığının kontrolünü yapan metotları tanımladım. *Person.hpp* dosyasında; kişinin isim, doğum tarihi, kilo, yaş gibi özellikleri ile bunlara ait getter/setter metotları ve 2 farklı constructor metodunu deklare ettim. AVL Ağacı ile ilgili işlemlerin yapıldığı dosyada (*AVLTree.hpp*), ekleme, silme, yüksekliğini bulma, tekli sola ve sağa dönüş, çiftli sola ve sağa dönüş, seviye alma ve güncelleme, Stack'e ekleme, postorder sıralama ve kök durumunu dönme gibi islemleri yapan metotları tanımladım. Programın çalışması için çeşitli kontrolleri yaptığım dosyada *(ProgramController.hpp*) ise, Programı çalıştıracak metot ile AVL ağacı, file stream ve kişinin ad, doğum tarihi, kilo bilgilerinin okunması için gerekli değişkenlerin tanımlanmasını icra ettim. Böylelikle daha düzenli bir kod yapısı hedefledim ve bu dosyaları main bloğumuzun bulunduğu dosyaya dahil edip işlemleri gerçekleştirdim.

Projeyi oluştururken düğümlerin yer değiştirdikleri zaman stack yapısına gerekli harfleri atamak için uygun olacak algoritmayı bulmak beni zorlayan kısımdı. Bu algoritmayı çözmek için gerekli araştırmalar yaptım ve kâğıt üzerinde eklemeler yaparak gözlemlediğim değişikliklere göre bir algoritma ortaya çıkardım. Bu süreç içerisinde işaretleyicilerle çok uğraştığım için hafıza yönetimi konusunda ödevin bana fayda sağladığını düşünüyorum. Proje farklı girdilerle test ettim ve kağıt üzerinde elle deneyerek kontrol ettim. Kontrolün sonucunda proje farklı sayıda girdilerle de doğru çıktıyı veriyor. Kaynak dosyalarında karmaşık fonksiyonların ne işe yaradığını yorum olarak belirtmeler halinde yaptım. Son olarak mingw yardımı ile yazılan bütün kodları derledim ve uygun bir makefile yazdım. Makefile yazımı için internetten birkaç video izledim ve gerekli klasör hiyerarşisini oluşturdum. Beni oku dosyasında projenin cmd üzerinde nasıl çalıştırılacağı açıklanmıştır.

KAYNAKÇALAR

[1] https://www.geeksforgeeks.org/avl-tree-set-1-insertion/
[2] https://www.softwaretestinghelp.com/stack-in-cpp/#:~:text=All%20That%20You%20Need%20To,which%20the%20operations%20are%20performed.
[3] http://pages.cs.wisc.edu/~ealexand/cs367/NOTES/AVL-Trees/index.html

Not: İşbu rapor şahsıma ait olup, yararlandığım kaynaklar yukarıdaki gibidir. Kimseyle paylaşmadığımı belirtmek isterim.