FATMA GÜNER

**VERİ YAPILARI VE ALGORİTMALARI 2. ÖDEV RAPORU**

Bu projeyi gerçekleştirmek için gerekli adımları belirledim ve ardından sırayla kodu yazdım;

Adım Adım Yaptıklarım:

1. Dosya, klasör ve dizin yapısını belirleme ve gerçekleme işlemini gerçekleştirdim ilk önce

(include/ ‘Konum.hpp’ ’Labirent.hpp’ ‘Stack.hpp’

src/ ‘Konum.cpp’ ‘Labirent.cpp’ ‘Nokta.cpp’ ‘main.cpp’

lib/

bin/ )

2. İlk önce Konum.hpp başlık dosyasında yönleri (aşağı, sol, yukarı, sağ) ve konum nesnesi oluşturmak için kullanılan kurucu fonksiyonlar tanımlanır. Bu başlık dosyası, Konum yapısının ve onunla ilgili işlevlerin tanımlarını içeren bir arayüz sağlar. Labirent problemini çözerken, Konum yapısını kullanarak labirentteki hücrelerin koordinatlarını ve yönlerini kolayca yönetebiliriz.

3. Konum.cpp dosyasını oluşturdum. Konum sınıfı, bir labirentteki konumları ve yönleri temsil eder ve bu konumlarla ilgili çeşitli işlemleri sağlar. Öncelikle burada kurucu fonksiyonu oluşturdum ve bu kurucu fonksiyon, x ve y koordinatları ve bir yön (yon) alır ve bu değerleri Konum nesnesinin üyelerine atar. Bu kurucu fonksiyon, sadece x ve y koordinatlarını alır ve yönü varsayılan olarak ASAGI (aşağı) olarak ayarlar. Daha sonra yön değiştirme ve hareket işlevlerini belirledim. Ayrıca geri izleme (backtracking) sırasında kullanışlı olması açısından ters yönü belirleme işlevi mevcut yönün ters yönünü döndürür. Örneğin, mevcut yön ASAGI ise ters yön YUKARI olur.

Bu yapılan işlevlerin tümü, bir labirentte gezinmeyi ve yönleri yönetmeyi kolaylaştırmak için tasarlanmıştır. Her bir işlev, Konum nesnesinin mevcut durumunu ve yönünü kullanarak yeni konumlar veya yönler hesaplar ve döndürür.

4. Harita.txt dosyasına labirenti yazdıktan sonra Labirent.hpp başlık dosyasını yazdım. Bu sınıf, labirentin yapısını, başlangıç ve bitiş konumlarını, yönleri ve labirenti çözmek için gerekli fonksiyonları içerir. Gerekli kütüphaneleri dahil ettikten sonra labirentin yüksekliği ve genişliği için sabit değerleri tanımladım. Labirent sınıfını tanımladım, bu sınıf labirent ile ilgili tüm verileri ve işlevleri içerir;

char harita[YUKSEKLIK][GENISLIK]: Labirentin karakter matrisini tutar .

int x, y: Mevcut konumun koordinatlarını tutar.

Yon yon: Mevcut yönü tutar.

Konum bitis: Bitiş konumunu tutar.

Stack<Konum> \*yigit: Konumları tutan bir yığın nesnesini işaret eder.

Bu sınıf, bir labirent problemini çözmek için gerekli tüm verileri ve işlevleri içeren bir arayüz sağlar. Labirentin yapısını oluşturur, hareket eder, engelleri kontrol eder ve labirentin çözümünü sağlar.

5. ardından Labirent.cpp dosyasını oluşturdum. Bu sınıf, labirentin okunması, hareket edilmesi ve labirentin çözülmesi için gerekli fonksiyonları içerir. Yazılan kodda kurucu fonksiyon, Harita.txt dosyasından labirenti okur ve harita matrisine kaydeder. Başlangıç (x, y) ve bitiş (bitis) konumlarını ayarlar. Ayrıca, yönü ASAGI (aşağı) olarak başlatır ve yigit yığınına bir başlangıç değeri ekler. Adım atma fonksiyonu, mevcut konumdan ileri konuma geçmeye çalışır. Eğer ileri konumda engel yoksa, mevcut konumu yığına ekler ve ileri konuma geçer. Engel varsa false döner. Ayarla fonksiyonu, mevcut konumu ve yönü günceller, haritayı günceller ve ardından haritayı ekrana yazdırır. Ekranı temizlemek için system("cls") komutu kullanılır ve kısa bir bekleme süresi eklenir (Sleep(30)). CikisaGeldimi fonksiyonu, mevcut konumun bitiş konumuna eşit olup olmadığını kontrol eder. Eğer eşitse true, değilse false döner. EngelVarmi fonksiyonu, verilen konumun harita sınırları içinde olup olmadığını ve o konumda bir engel (#) veya işaretli bir yol (-) olup olmadığını kontrol eder. Engel varsa true, yoksa false döner. Labirenti yazdıran fonksiyon, harita matrisini ve mevcut konumu (yönü ile birlikte) yazdırır. Mevcut konumu, yön karakteri (v, <, ^, >) ile gösterir.

Yani kısaca bu sınıf, bir labirent problemini çözmek için gerekli işlevselliği sağlar. Labirent sınıfı, dosyadan labirenti okur, mevcut konumu ve yönü yönetir, adımlar atar ve çıkışa ulaşana kadar labirentte gezinir.

6. Nokta.cpp dosyasını oluşturdum. Bu dosyada genel olarak, Nokta (veya Point gibi düşünebiliriz) adında bir sınıfın implementasyonunu içerir. Nokta sınıfı, iki boyutlu bir noktayı temsil eden temel işlevleri sağlar. Bu kod, bir nokta sınıfını tanımlar ve implement eder. Nokta sınıfı, x ve y koordinatlarını saklar ve bu koordinatlarla ilgili temel işlemleri (getter ve setter) sağlar. Ayrıca, sınıfın nesnelerinin kolayca ekrana yazdırılabilmesi için bir akış operatörü aşırı yüklemesi içerir.

7. Stack.hpp başlık sınıfını yaptım. Bu kod, genel bir yığın (stack) veri yapısını implement eden bir Stack sınıfı ve yardımcı bir Node sınıfını tanımlar. Node sınıfı, yığındaki her bir öğeyi temsil ederken, Stack sınıfı ise bu düğümleri yönetir. Bu kod, yığın (stack) veri yapısını implemente eder ve temel yığın işlemlerini (push, pop, top, isEmpty) sağlar. Ayrıca, makeEmpty fonksiyonu, yığını temizler ve destrüktör, bellek yönetimini sağlar.

8. main.cpp dosyasını oluşturdum. Bu kod, bir labirentte DFS algoritması kullanarak bir çözüm yolu bulmaya çalışır. Başlangıç noktasından itibaren, her adımda ilerlerken, uygun bir yol bulana kadar farklı yönleri dener. Eğer ilerleme mümkün değilse, geri dönüş yaparak alternatif yolları denemeye devam eder. Labirent çözümü başarıyla tamamlandığında, kullanıcıya mesaj verilir ve program sona erer.