

**Dr. Öğr. Üyesi Selman HIZAL**

**Veri Yapıları ve Algoritmalar dersi**

**Teknoloji Fakültesi**

**Bilgisayar Mühendisliği 2.sınıf**

**OgrenciNo = “B210109048”**

**AdSoyad = “Furkan Duran”**

**Stack ile Labirent Çıkışı Bulma**

**1. Node Sınıfı:**

-Node sınıfı ile bağlı liste oluşturmuş oluruz . yani bir tane verimiz var ve bu verimiz pointer yardımı ile farkli bir veriye işaret ederek ona bağlanıyor .

- node ile yığının düğümleri temsil edilerek veriyi saklamaya yarar.

Node sınıfının UML kodu :

metin, ekran görüntüsü, iş kartı, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**2. Stack Sınıfı:**

-Stack(yığıt) yapısı LIFO (last in first out) ilkesini izleyen bir doğrusal veri yapısıdır.

- Yığıt, labirent çözümü sırasında ziyaret edilen konumları ve bu konumları takip etmek için kullanılır. Peki neden ? Çünkü yığından geri dönüş imkanı vardır.

- **isEmpty** : Yığıt boş mu kontrolu işlemi

**-push** : yığıta ekleme işlemi

-**pop** : yığıttan eleman silme işlemi

-**top** : yığıta son giren eleman

-**makeEmpty :** yığıtı boşaltma işlemi(yığıtın artık kullanılmadığı bir noktada bellekteki tüm düğümleri serbest bırakarak bellek sızıntılarını önlemek için kullanılabilir.

Stack sınıfının UML kodu :

metin, ekran görüntüsü, yazı tipi, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3. Konum ve Yön Yapıları:**

- Konum yapısı, labirentteki bir konumu (x, y koordinatları) ve bir yönü, (ASAGI, SOL, YUKARI, SAG) temsil için kullanılır.

- Yon enum'u, konumu belirli bir yönde güncellemek için kullanılır.

**-Konum(int** x**, int** y**, Yon** yon**):**

Bu yapıcı fonksiyonda belirli bir koordinatta (x,y) ve belirli bir yönde(ASAGI,YUKARI,SAG,SOL) bulunan konumu temsil eder.

**-Konum(int** x**, int** y**):**

Bu yapıcı fonksiyonda ise belirli bir koordinatta (x,y) ve başlangıçta varsayılan olarak aşağı yönde (ASAGI) bulunan bir konumu temsil eder.

**-Konum():**

Bu yapıcı fonksiyonda ise başlangıç değeri için (0,0) ve yön için de ASAGI varsayılanları konumu temsil eder , yani kıssaca default değerlerimiz diyebiliriz.

**-Konum AyniYon():**

Mevcut `Konum` nesnesinin bulunduğu yönde yeni bir konum oluşturur. Örneğin, eğer mevcut konumun yönü `YUKARI` ise `Yukari()` fonksiyonu ile aynı yönde yeni bir konum oluşturulur.

**-Konum SaatYonu(Yon** yn**):**

Saat yönünün tersine yeni bir konum oluşturur. Belirli bir yön (YUKARI, SAG, ASAGI, SOL) verildiğinde, bu yönün tersine yeni bir konum oluşturulur.

**-Konum Asagi():** Mevcut konumun aşağısında bulunan yeni bir konum oluşturur.

**-Konum Yukari():**Mevcut konumun yukarısında bulunan yeni bir konum oluşturur.

**-Konum Sol():** Mevcut konumun solunda bulunan yeni bir konum oluşturur.

**-Konum Sag():** Mevcut konumun sağında bulunan yeni bir konum oluşturur.

**-Yon TersYon():**Mevcut konumun ters yönünü döndürür.

Labirenti çözerken mevcut konumu güncellemek ve ilerlemeleri kaydetmek yani yeni konumları oluşturmak için bu sınıf bize yardımcı olur.

ekran görüntüsü, metin, yazı tipi, siyah içeren bir resim

Açıklama otomatik olarak oluşturulduKonum sınıfının UML kodu :

**4.Labirent Sınıfı:**

Labirent sınıfı, labirenti temsil eder ve çözüm burada gerçekleşir.

- **Labirent(Konum** baslangic**, Konum** bitis**):**

Bu fonksiyoun labirent classından bir tane nesne oluşturur ve baslangic parametresi labirenteki baslangic konumunu , bitis ise labirentten çıkış yani çözüme ulaşacağımız yerin konumunu temsil eder. Burada dışarıdan harita.txt olan bir metin belgesi içerisindeki haritamız **<fstream>** kütüphanesinin **<ifstream>** methodu ile oluşturduğumuz harita dizisine harita dosyasından okunan değerleri atar.

Bir stack nesnesi oluşturur ve başlangıç yönünü yığına ekler.

**-Konum Mevcutkonum():** mevcut konumu return ettiğimiz fonksiyon.

**-bool adimAt(Konum** mevcut**, Konum** ileri**)**

mevcut konumu ve ileri konumunu parametre olarak alır. Eğer `ileri` konumunda engel yoksa, mevcut konumu bir Stack içine ekler. Sonra ayarla fonksiyonunu çağırıp konumu ve yönü günceller.

Başarılı bir adım atıldıysa `true`, engel varsa `false` döndürür.

**-void ayarla(Konum** konum**, Yon** yon**)**

Ekrana labirenti yazdırdığımız kısım bu kısımdır. system("cls") ile ekrandaki yazıların hepsi silinir ve sadece labirentimiz kalır.

Konumu ve yönü güncelleyerek, `harita` dizisine `█` (**char**(219)) ekler. Ve yaz() fonksiyonu yardımı ile ekrana yazdırılmış olur.

**-bool** CikisaGeldimi()

Çıkış için girilen konumun kontrolunu gerçekleştirir eğer çıkışa gelindi ise true değer döndürür değilse false .

**-bool** EngelVarmi(**Konum** konum)

Gelen konum nesnesinin labirent sınırları içinde ve engel olup olmadığını kontrol eder. Bir nevi labirent sınırlarını programımıza tanıtmış oluyoruz.

Eğer konum labirent sınırları içinde değilse “veya” konumda bir engel varsa true , yoksa false boolian değer döndürmesini sağlar.

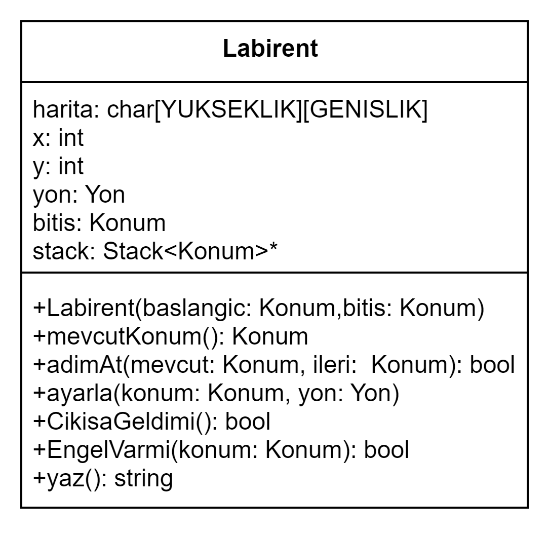
**-string** yaz()

Labirenti ekrana yazdırmak için kullanılan fonksiyondur.

Mevcut konumun yönünü gösteren ok karakterini (`^`, `<`, `v`, `>`) kullanarak labirenti ekrana basar. *not =Normalde charYon[]=(31,16,30,17) ; kullanacaktım .ama bu kodda çalıştıramadığım için alternatif semboller kullandım.*

Diğer karakterleri ve ziyaret edilen konumları █ olarak gösterir.

Bu fonksiyonlar, labirenti çözmek ve adımları ekrana yazdırmak için kullanılır. Özellikle adimAt ve ayarla fonksiyonları, labirentteki adımları takip etmek ve ekrana güncel labirent durumunu yazdırmak için önemlidir. `CikisaGeldimi` fonksiyonu ise çıkışa ulaşılıp ulaşılmadığını kontrol eder.

Labirent sınıfının UML kodu :

**5-Main() Kısmı**

Main kısmı her kodda yazdığımız gibi burada da yazıyoruz ve oluşturacağımız nesneleri burada tanımlayıp çağırmaları yapıyoruz.

İlk olarak labirent nesnesi oluştururuz ve başlangıç ve çıkış konumlarını belirleriz.

Başlangıç konumu stack içerisine ekleriz . ayarla fonksiyonu kullanarak başlangıç konumu ekrana yazdırır.

Labirenti çözmek için bir döngüye ihtiyacımız var.

*while (!labirent->CikisaGeldimi()) {*

*// ...*

*}*

Bu döngü, labirentin çözümünü gerçekleştirir. Çıkışa ulaşıldığında (`CikisaGeldimi()` fonksiyonu true döndüğünde) döngüden çıkılır.

Adım atma ve yön değiştirme işlemleri ise

if (!labirent->adimAt(mevcutKonum, ileri)) {

// ...

}

Bu blok, mevcut konumdan ileriye adım atılıp atılamayacağını kontrol eder. Eğer adım atılamıyorsa, farklı yönlere dönerek deneme yapar.

Yönlere dönme ve yön belirleme işlemleri ise

int i = 0;

bool sonuc = false;

Konum yeni = mevcutKonum;

while (!sonuc && denemeYonSayisi < 5) {

yeni = mevcutKonum.SaatYonu((Yon)((mevcutKonum.yon + i) % 4));

i++;

denemeYonSayisi++;

if (yeni.yon == mevcutKonum.TersYon()) {

sonuc = false;

}

else {

sonuc = labirent->adimAt(mevcutKonum, yeni);

}

}

Burası , mevcut yönden farklı yönlere dönerek deneme yapar. Eğer bir yönde adım atılabiliyorsa, adım atılır.

Stack'ten önceki konuma dönme işlemi ise aşağıdaki kod bloğu ile sağlanır

if (denemeYonSayisi == 5) {

labirent->stack->pop();

labirent->ayarla(oncekiKonum, oncekiKonum.TersYon());

}

Eğer 5 yönde de adım atılamazsa, stack'ten bir önceki konuma gidilir ve ekran yeniden ayarlanarak geri dönüş gösterilir.

Programın sonunda, çıkışa ulaşıldığını belirten bir mesaj yazılır ve program sonlandırılır.

Ozet olarak kısaca ;

- Labirent, bir .txt dosyasından okunur.

- Labirent çözülürken, mevcut konum ve yön bilgileri stack ile yapılmış olur.

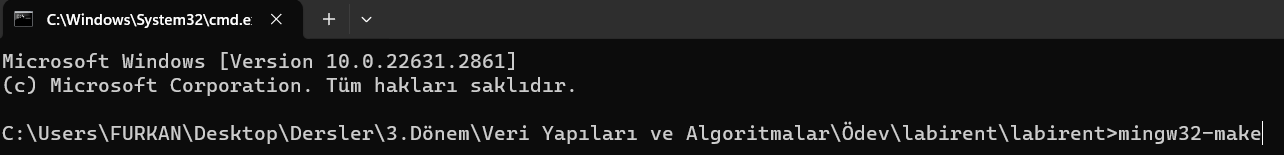
**Makefile dosyası oluşturma ve klasör düzeni**

programların kaynak dosyalarının birbirleri ile bağımlılıklarını derleme ve inşa etme aşamasında yönetmek için kullanılır. Makefile dosyasının uzantısı yoktur .

**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

kullanılmak istenen klasörde cmd açılır ve mingw32-make komutu ile çalıştırılmış olur.

****

**Ve Konsolda Uygulamamızı Başlatmış Oluruz.**

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

*Yardım Aldığım Kaynaklar :*

*Kendi ders notlarımız , chatGPT , https://gokhansengun.com/makefile-ve-make-nedir-ne-ise-yarar/*

*youtube : https://www.youtube.com/watch?v=mTdhPXfsU8c*

*https://youtu.be/SGqmFMm9eEo?si=I7b6WcMXpLCNjTeP*