

**AD:** HAFİZE CEMİLE

**SOYAD:** ÖĞÜT

**DERS:** VERİ YAPILARI VE ALGORİTMALAR

**NUMARA:** B210109058

**ÖDEV:**LABİRENT(MAZE) OYUNU:BACKTRAKING UYGULAMASI

**AKADEMİSYEN:**Dr.Öğr.Üyesi Selman HIZAL & Arş.Gör Emin GÜNEY

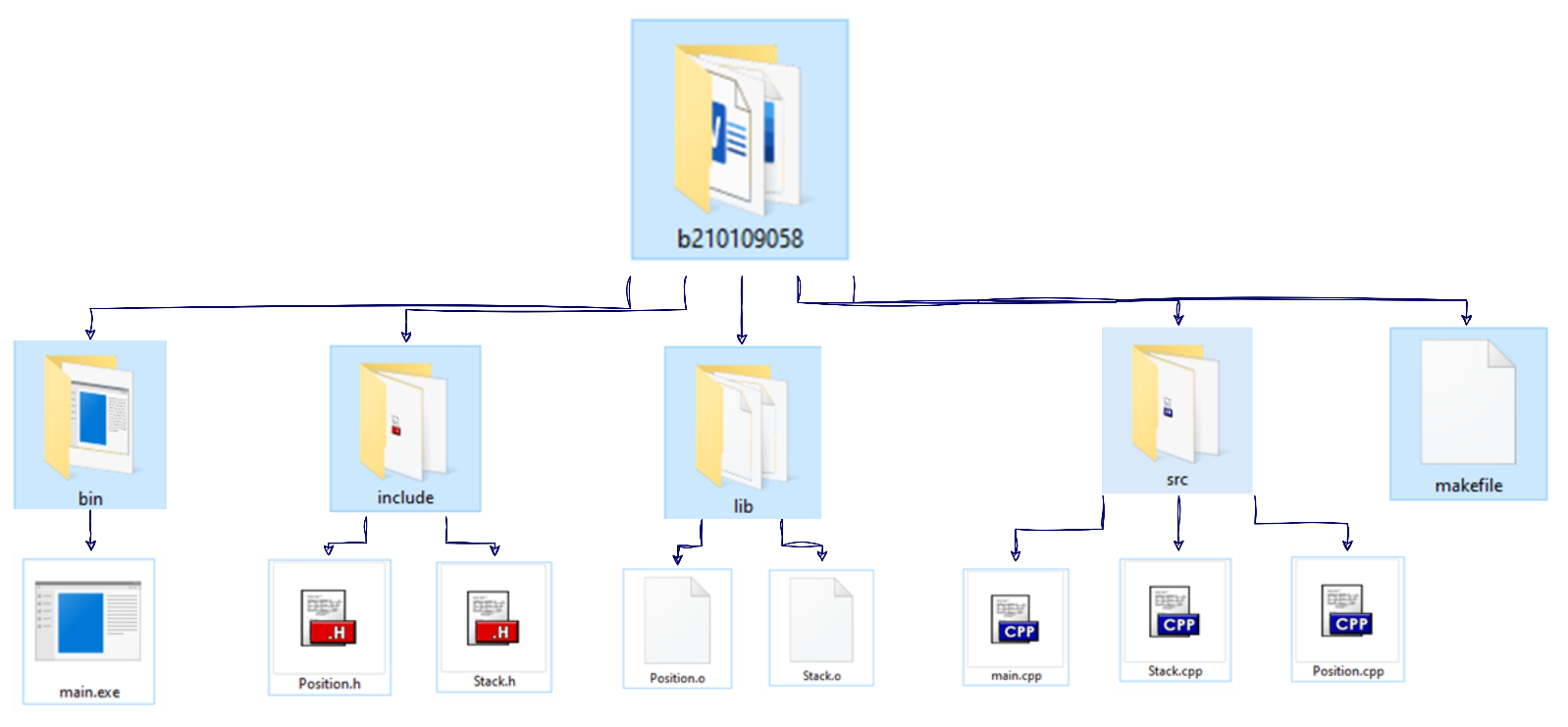
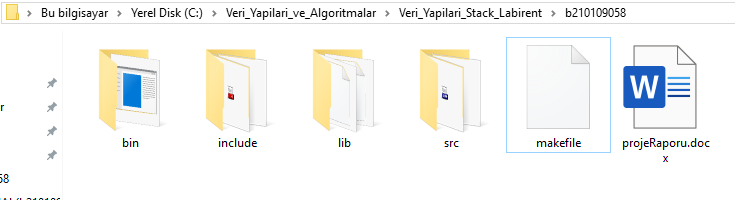
**Ben Kimim?**

Merhabalar, ben Hafize Cemile Öğüt. 15 Mayıs 2000 tarihinde Uşak’ta doğdum. İlk ve orta öğretim hayatımı Uşak’ta tamamladım. Liseyi Uşak Mesleki ve Teknik Anadolu Lisesi, Bilişim /Web Teknolojileri bölümünden mezun oldum. Öğrenimime bilgisayar mühendisliği öğrencisi olarak Sakarya’ da devam ediyorum.

**Bu Ödevdeki Algoritmam Nedir?**

1. Bir Labirent, blokların N\*M ikili matrisi olarak verilir ve başlangıçta (0,0) yani bir \* simgesi vardır.
2. Labirent [0][0] ve \* labirentteki belirli bir blokta bulunan yiyecekleri yemek ister.
3. Bir labirent matrisinde #, bloğun bir çıkmaz sokak olduğu anlamına gelir ve boşluk karakterleri, bloğun kaynaktan hedefe giden yolda kullanılabileceği anlamına gelir.
4. Karakteri, bloğun çıkmaz bir yol olmaması koşuluyla herhangi bir bloğa herhangi bir yönde (çapraz olarak değil) hareket edebilir.
5. Görev, \* karakterini yiyeceğe ulaşıp ulaşmayacağı herhangi bir yol olup olmadığını kontrol etmektir.
6. Yolu yazdırmak için gerekli değildir.
7. Başlangıçta gerekli dizinlerine sahip yığın yapısı oluşturup, yolları yığına iteceğiz.
8. En üstteki düğümün tüm yönlerine saat yönünün tersine bir şekilde birer birer hareket edeceğiz ve her seferinde yeni bir yol denediğimizde bu düğümü (labirentin bloğu) yığına iteceğiz.
9. Her seferinde en üstteki düğümün gerekli değişkenini arttıracağız, böylece tüm yönler araştırılmadıkça her seferinde yeni bir yön deneyebiliriz.
10. Ayrıca, labirenti hangi bloklarının yolda zaten kullanıldığını veya başka bir deyişle yığında mevcut olduğunu koruyacak ziyaret edilen bir matrisi de koruyacağız.
11. Herhangi bir yönü denerken, labirentin bloğunun çıkmaz bir yol olup olmadığını ve labirentin dışında olup olmadığını da kontrol edeceğiz.
12. Bunu, ya en üstteki düğüm koordinatları yiyeceğin koordinatlarına eşit hale gelirken yapacağız, bu da yiyeceğe ulaştığımız anlamına gelir ya da yığın boşalır, bu da yiyeceğe ulaşmanın olası bir yolu olmadığı anlamına gelir.

**PROGRAM TANITIMI:**

Öncelikle LMS sistemine yükleyeceğim dosyayı göstermek istiyorum. 

Resimde de gördüğünüz gibi, bin, include,lib,src adlarınla yeni klasörler oluşturdum.

Src klasörümde cpp uzantılı main, pointer ve stack dosyalarım var.

İnclude klasörümde ise başlık uzantılı (.h) dosyalarım var(Position.h, Stack.h).

Daha sonra, makefile dosyamı vsCode açıp gerekli komut satırlarını yazdım;

metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ve gelelim programın çalışmasına, gerekli komutlarla (mingw32-make) cmd ekranında programımı çalıştırıyorum

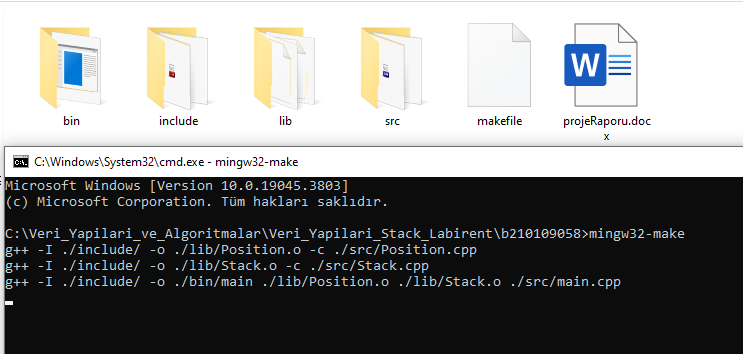
metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu işlem için bilgisayarımızda mingw kurulu olmalıdır. Aynı zamanda gerekli dizinde işlem yapmalısınız.

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

Açıklama otomatik olarak oluşturuldu



Ve programımın çalışma videosuna da aşağıdaki dosyaya çift tıklayarak izleyebilirsiniz.

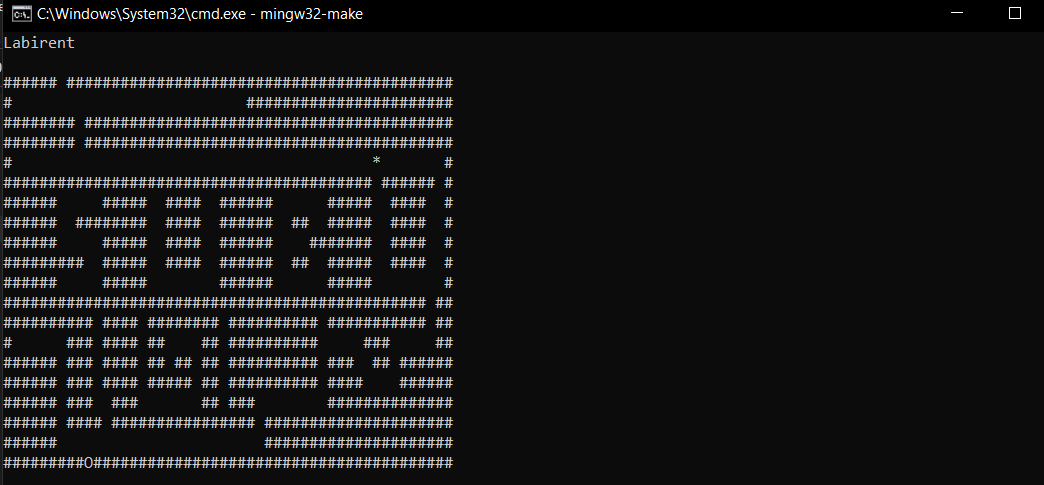


Veya <https://drive.google.com/drive/folders/1lUq780OUUQA8ZtWiwYjkGfSdHTi0ZQ4r?usp=drive_link> bu linkten ulaşabilirsiniz.

Yalnızca benim yaptığım sade haritayla kalmayıp, kaynak (main) kodunun içinde gerekli labirent haritanın satır ve sütun sayılarını değiştirerek kendi haritanızı oluşturabilirsiniz.

Örnekte kendi girdiğim bir harita var ve labirent bunu çözüyor

Kaynak kodlarımda bu haritaya erişebilirsiniz.



Algoritma ve Kodlarımı detaylı inceleyecek olursak:

Header dosya kullanımı: Bu başlık dosyaları kullanılan tanımlamaları içerir.

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

Açıklama otomatik olarak oluşturuldu

#ifndef POSITION\_H //Sembolün daha önce tanımlanmamış olması durumunu kontrol eder.

#define POSITION\_H

// Position\_H sembolü daha önce tanımlanmamışsa (ifndef), aşağıdaki blok çalışır

// Bu, başlık dosyasının tekrarlı dahil edilmemesini sağlar //

struct Position

{

    int x;  // X koordinatı

    int y;  // Y koordinatı

};

#endif

metin, dikdörtgen, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu1.Başlatma:

1. Stack : stack sınıfının yapıcı bir fonksiyonu çalıştır.

2. temel adlı dinamik bir dizi oluştur.

3. top değeri başlangıçta -1 olarak ayarlanır.

4. length başlangıçta 0 olarak ayarlanır.

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

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazı tipi, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

2.Ekleme:

1. Eğer top yığının sonuna ulaştıysa yığın doludur ve 1 döndür.

2. Top artırtılır ve yeni eleman temel dizisine eklenir.

3. top ve length arttırılır. elem diziye alınır

metin, el yazısı, yazı tipi, Post-it notu içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, el yazısı, mektup, harf, Post-it notu içeren bir resim

Açıklama otomatik olarak oluşturuldu3.Çıkarma:

1. x ve y koordinatları belirlendi
2. Eğer top=-1 ise (yığın boştur) Position yapısını döndür
3. eğer top=-1 ise error döndür.
4. top elemanı alınır,top azaltılır,lenght arttırılır.

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

Açıklama otomatik olarak oluşturuldu4.Eleman Sayısını Alma

1. eleman sayısını döndür.

#pragma once

#include "Position.h"

// Stack sınıfı, Position (Pozisyon) veri yapısını kullanarak bir yığıın (stack) implementasyonunu sağlar.

class Stack

{

private:

    Position\* temel; // Yığının temelini tutan dinamik dizi

    int top;         // Yığının üstündeki elemanın indeksi

    int size;        // Yığının toplam kapasitesi

    int length;      // Yığındaki mevcut eleman sayısı

public:

    // Yapıcı (constructor) fonksiyon, yığının başlangıç durumunu oluşturur.

    Stack(int capacity)

    {

        temel = new Position[capacity]; // Yığına ait dinamik dizi oluşturulur

        size = capacity;                // Yığının kapasitesi belirlenir

        top = -1;                       // Yığının başlangıçta boş olması için top -1 olarak ayarlanır

        length = 0;                     // Yığındaki eleman sayısı sıfıra ayarlanır

    }

    // Yıkıcı (destructor) fonksiyon, dinamik belleği serbest bırakır.

    ~Stack()

    {

        delete[] temel; // Yığına ait dinamik diziyi bellekten sil

        top = -1;        // Yığındaki üst eleman indeksini sıfırla

        size = 0;        // Yığının kapasitesini sıfırla

        length = 0;      // Yığındaki eleman sayısını sıfırla

    }

    // Yığına eleman ekleyen fonksiyon

    int Push(Position elem)

    {

        if (top == size - 1) return 1; // Yığın dolu. Başarısızlık durumunu 1 değeri ile döndür

        else

        {

            top++;          // Üst eleman indeksini bir artır

            temel[top] = elem; // Yığına yeni elemanı ekle

            length++;       // Yığındaki eleman sayısını bir artır

            return 0;        // Başarı durumu için 0 değeri ile döndür

        }

    }

    // Yığından eleman çeken fonksiyon

    Position Pop()

    {

        Position error;

        error.x = -1;

        error.y = -1;

        Position correct;

        if (top == -1) return error; // Yığın boş. Var olmayan bir pozisyonu döndür

        else

        {

            correct = temel[top]; // Doğru konumu sakla

            top--;               // Üst eleman indeksini bir azalt

            length--;            // Yığındaki eleman sayısını bir azalt

            return correct;      // Başarı durumu için doğru konumu döndür

        }

    }

    // Yığındaki eleman sayısını döndüren fonksiyon

    int GetLength()

    {

        return length;

    }

};

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

Açıklama otomatik olarak oluşturuldu

1. GIRIS = 'I',CIKIS = 'O',DUVAR = '#',YOL = ' ',KESIF = 'e';
2. Labirentin içeriğini giriniz
3. Labirentin iki kopyasını oluştur.
4. Başlangıç koordinatlarını belirle
5. i ve j değişkenleri ile oluşturulan matris üzerinde gezin
6. Labirent içindeki yolları takip için stack yapısı oluştur
7. Şu anki konumu labirent başlangıcına ata
8. Labirentte sürekli olarak dolaş
9. Eğer şu anki konum Çıkışa eşitse dur.
10. Yukarı, Sağ , Aşağı ve Sol koordinatları tespit et
11. a=1
12. While -> Eğer a=1 ise

1.Eğer Yukarı pozisyonu YOL a eşitse veya Yukarı pozisyonu CIKIS a eşitse

* + - 1. Şu anki pozisyonu stack e ekle
      2. Şu anki pozisyon KESIF e eşitle

3.Mevcut konumun x koordinatı 1 azaltılır. Bir üst satıra geçilir.

2.Sağ pozisyonu YOL a eşitse veya sağ pozisyonu CIKIS a eşitse

1.Şu anki pozisyonu stack e ekle

2.Şu anki pozisyon KESIF e eşitle

3.Şu anki pozisyonu 1 artır. Y koordinatını 1 arttır

3.Aşağı pozisyonu YOL a eşitse veya Aşağı Pozisyon CIKIS a eşitse

1.Mevcut konumu stack e ekle

2.Şu anki pozisyonu KESIF e eşitle

3.x koordinatını artır. Bir alt satıra geç

14. Şu anki pozisyonu KESIF e eşitle

15. Boyut belirlenir

16.i =0 i’yi arttır. Eğer i < son boyuttan

1.Karakterin şuanda bulunduğu konumu \* karakteri ile işretle

2.a =0 a’yı arttır. Eğer a< SATIR boyuttan

1.b =0 b’yiı arttır. Eğer a< SUTUN boyuttan

1. Şu anki konumu yazdır.

2. Git 16.2.1. Adıma

2. Git 16.5. Adıma

3. Karakterin şuanda bulunduğu konumu ‘ ‘ karakteri ile işaretle

4. Git 16. Adıma

17. a =0 a’yı arttır. Eğer a< SATIR boyuttan

1.b =0 b’yiı arttır. Eğer a< SUTUN boyuttan

1. Şu anki konumu yazdır.

2. Git 17.1.1. Adıma

18. Çıkışa Gelindi Yaz.

metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ana klasörde bulunan  bu dosyayı tarayıcınızda https://www.drawio.com/ gidip sürüklediğinizde veya bu word dosyasını yakınlaştırdığınızda akış şemasını detaylı bir şekilde inceleyebilirsiniz.

//Ana Fonksiyon

#include <iostream>

#include <fstream>

#include <windows.h>

#include "Stack.h"

#include "Position.h"

#include <vector>

#define SATIR 21

#define SUTUN 51

using namespace std;

int main()

{

    const char GIRIS = 'I';

    const char CIKIS = 'O';

    const char DUVAR = '#';

    const char YOL = ' ';

    const char KESIF = 'e';

    // Labirentin içeriğini girildi

    char labirent[SATIR][SUTUN] = {

        // Labirent haritası buraya eklenmelidir

    };

    // Lütfen haritayı girerken satır ve sütun sayılarını 1 eksik gir

    Position labirentGiris;

    labirentGiris.x = 0; // Labirentin başlangıç koordinatları

    labirentGiris.y = 6;

    // Labirentin iki kopyası oluştur

    char labirentKopya[SATIR][SUTUN];

    char labirentGoruntule[SATIR][SUTUN];

    for (int i = 0; i < SATIR; i++)

    {

        for (int j = 0; j < SUTUN; j++)

        {

             // labirentGoruntule matrisine, labirent matrisinin içeriğini ata

            labirentGoruntule[i][j] = labirent[i][j];

             // labirentKopya matrisine, labirent matrisinin içeriğini kopyala

            labirentKopya[i][j] = labirent[i][j];

        }

    }

    // Lbirent içindeki yolları TAKİP İÇİN Bir yığın (stack) veri yapısı OLUSTUR

    Stack path(SATIR \* SUTUN);

    Position suankiPozisyon;

    // SuankiPozisyon değişkenine, labirentin başlangıç koordinatları ata

    suankiPozisyon = labirentGiris;

    while (true)

    {

        if (labirentKopya[suankiPozisyon.x][suankiPozisyon.y] == CIKIS)

            break;

        // Mevcut konumun her yöndeki labirent öğesini al

        char pozisyonYukari = labirentKopya[suankiPozisyon.x - 1][suankiPozisyon.y];

        char pozisyonSaga = labirentKopya[suankiPozisyon.x][suankiPozisyon.y + 1];

        char pozisyonAsagi = labirentKopya[suankiPozisyon.x + 1][suankiPozisyon.y];

        char pozisyonSola = labirentKopya[suankiPozisyon.x][suankiPozisyon.y - 1];

        // Bu pozisyonu KESIF olarak işaretle, sonra bir pozisyon yukarı git

        if ((pozisyonYukari == YOL || pozisyonYukari == CIKIS) && suankiPozisyon.x - 1 >= 0)

        {

            path.Push(suankiPozisyon);

            labirentKopya[suankiPozisyon.x][suankiPozisyon.y] = KESIF;

            suankiPozisyon.x--;

        }

        // Bu pozisyonu KESIF olarak işaretle, sonra bir pozisyon sağa git

        else if ((pozisyonSaga == YOL || pozisyonSaga == CIKIS) && suankiPozisyon.y + 1 < SUTUN)

        {

            path.Push(suankiPozisyon);

            labirentKopya[suankiPozisyon.x][suankiPozisyon.y] = KESIF;

            suankiPozisyon.y++;

        }

        // Bu pozisyonu KESIF olarak işaretle, sonra bir pozisyon aşağı git

        else if ((pozisyonAsagi == YOL || pozisyonAsagi == CIKIS) && suankiPozisyon.x + 1 < SATIR)

        {

            path.Push(suankiPozisyon);

            labirentKopya[suankiPozisyon.x][suankiPozisyon.y] = KESIF;

            suankiPozisyon.x++;

        }

        // Bu pozisyonu KESIF olarak işaretle, sonra bir pozisyon sola git

        else

        {

            labirentKopya[suankiPozisyon.x][suankiPozisyon.y] = DUVAR;

            suankiPozisyon = path.Pop();

        }

    }

int sonBoyut = path.GetLength(); // Yığının uzunluğunu alır

Position \*sagPath = new Position[SATIR \* SUTUN]; // SATIR ve SUTUN değerlerine göre yeni bir pozisyon dizisi oluşturur

// Yığından elemanları ters sırayla sagPath dizisine atar

for (int i = sonBoyut - 1; i >= 0; i--)

{

    sagPath[i] = path.Pop(); // Yığından elemanları çıkarıp sagPath dizisine atar

}

// sagPath'teki her elemanı kullanarak labirenti ekranda görselleştirir

for (int i = 0; i < sonBoyut; i++)

{

    cout << "Labirent" << endl;

    cout << endl;

    Position geciciPozisyon = sagPath[i]; // Geçici olarak sagPath'ten bir pozisyon alır

    labirentGoruntule[geciciPozisyon.x][geciciPozisyon.y] = '\*'; // Labirent haritasında belirtilen pozisyona '\*' karakteri ekler

    // Labirentin güncellenmiş halini ekrana basar

    for (int a = 0; a < SATIR; a++)

    {

        for (int b = 0; b < SUTUN; b++)

        {

            cout << labirentGoruntule[a][b]; // Labirenti ekrana basar

        }

        cout << endl;

    }

    Sleep(200); // Belirli bir süre bekler

    system("cls"); // Ekranı temizler

    labirentGoruntule[geciciPozisyon.x][geciciPozisyon.y] = ' '; // '\*' karakterini kaldırarak eski haline getirir

}

// Orijinal labirent haritasını ekrana basar

cout << "Orijinal Harita: " << endl;

for (int a = 0; a < SATIR; a++)

{

    for (int b = 0; b < SUTUN; b++)

    {

        cout << labirent[a][b]; // Orijinal labirent haritasını ekrana basar

    }

    cout << endl;

}

cout << "CIKISA GELINDI!!" << endl; // Çıkışa ulaşıldığını bildirir

system("pause"); // Programın kapanmasını bekler

return 0;



#include <iostream>

#include "Position.h"

using namespace std;



#include <iostream>

#include <Stack.h>

using namespace std;



all: compile run

compile:

g++ -I ./include/ -o ./lib/Position.o -c ./src/Position.cpp

g++ -I ./include/ -o ./lib/Stack.o -c ./src/Stack.cpp

g++ -I ./include/ -o ./bin/main ./lib/Position.o ./lib/Stack.o ./src/main.cpp

run:

./bin/main

**FAYDALANDIĞIM KAYNAKLAR:**

[www.stackoverflow.com](http://www.stackoverflow.com)

[www.geeksforgeeks.org](http://www.geeksforgeeks.org)

[SteverNote | Blog of Tony Stever](https://stevernote.com/)

[https://cplusplus.com](https://cplusplus.com/)

[www.chegg.com](http://www.chegg.com)

[www.github.com](http://www.github.com)

[www.daniweb.com](http://www.daniweb.com)