T.C

KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ

DOĞA BİLİMLERİ VE MÜHENDİSLİK FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİK PROGRAMI

ÖDEV KONUSU
GEOMETRİK PROBLEMLER

Hazırlayan

Ahmet Can BOSTANCI – 220501031

https://github.com/bozokhalat

Nazlı Su KETÇİ – 220501007

https://github.com/nzlktc02

DERS SORUMLUSU

Prof.Dr.Hüseyin Tarık DURU

Yeniköy Mahallesi Ilıca Caddesi No:29, Başiskele,Kocaeli
info@kocaelisaglik.edu.tr kocaelisaglik.edu.tr

İÇİNDEKİLER

```cpp
#include <iostream>
#include <string>
#include <cmath>
#include "AllClass.h"
using namespace std;
using std::pow;
using std::tan;
using std::acos;
```

- **#include <iostream>**: Bu, giriş/çıkış akışı kütüphanesini içerir ve giriş ve çıkış işlemleri için gereklidir.
- **#include <string>**: Bu, string işlemleri ve operasyonları için string kütüphanesini içerir.
- **#include <cmath>**: Bu, matematik fonksiyonlarını içeren cmath (mathematics) kütüphanesini içerir; örneğin pow (üs alma), tan (tanjant), acos (arkkosinüs) gibi.
- **#include "AllClass.h"**: Bu, "AllClass.h" adlı kullanıcı tanımlı başlık dosyasını içerir. Bu dosyanın içeriği muhtemelen ana kod içinde kullanılan çeşitli sınıfların veya fonksiyonların deklarasyonlarını içerir.

- **using namespace std;**: Bu satır, **std** ad alanını geçerli kapsama (scope) getirir. Bu, **std::** ön eki olmadan standart C++ kütüphane bileşenlerini kullanmamıza olanak tanır.
- **using std::pow;**, **using std::tan;**, **using std::acos;**: Bu satırlar özellikle **std** ad alanından **pow**, **tan** ve **acos** fonksiyonlarını geçerli kapsama getirir. Bu, bu fonksiyonları ad alanını açıkça belirtmeden kullanmamıza olanak tanır.

  "AllClass.h" dosyasının ve geri kalan kodun içeriği olmadan programın tam işlevselliğini sağlamak mümkün değildir. Kod, matematiksel işlemleri içeren ve muhtemelen "AllClass.h" dosyasında bildirilen kullanıcı tanımlı sınıfları veya fonksiyonları içeren bir C++ programını kuruyormuş gibi görünüyor.

```cpp
Nokta::Nokta(const Nokta& Nesne) {
    x = Nesne.getX();
    y = Nesne.getY();
}

Nokta::Nokta(const Nokta& Nesne, double ofset_x, double ofset_y) {
    x = Nesne.getX() + ofset_x;
    y = Nesne.getY() + ofset_y;
}

Nokta::Nokta() {
    setX(0);
    setY(0);
}
```

```cpp
Nokta::Nokta(double xykordinat) {
    setX(xykordinat);
    setY(xykordinat);
}

Nokta::Nokta(double xkordinat, double ykordinat) {
    setX(xkordinat);
    setY(ykordinat);
}

void Nokta::setX(double xkordinat) {
    x = xkordinat;
}

void Nokta::setY(double ykordinat) {
    y = ykordinat;
}

void Nokta::set(double xkordinat, double ykordinat) {
    setX(xkordinat);
    setY(ykordinat);
}

double Nokta::getX() const {
    return x;
}

double Nokta::getY() const {
    return y;
}

std::string Nokta::toString() const {
    return "(" + std::to_string(x) + ", " + std::to_string(y) + ")";
}

void Nokta::yazdir() const {
    std::cout << "Koordinatlar: " << toString() << std::endl;
}
```

Fonksiyon Açıklamaları:

- **Constructor'lar:**
  - **Nokta(const Nokta& Nesne)**: Kopya oluşturma constructor'ı, başka bir Nokta nesnesini alır ve aynı koordinatları kullanarak yeni bir Nokta nesnesi oluşturur.
  - **Nokta(const Nokta& Nesne, double ofset_x, double ofset_y)**: Başka bir Nokta nesnesini ve ofset değerlerini alarak yeni bir Nokta nesnesi oluşturur.
  - **Nokta()**: Default constructor, x ve y koordinatlarını sıfır olarak ayarlar.
  - **Nokta(double xykordinat)**: Tek bir koordinat değeri alarak Nokta nesnesi oluşturur, hem x hem de y koordinatlarını bu değere ayarlar.
  - **Nokta(double xkordinat, double ykordinat)**: x ve y koordinat değerlerini

alarak Nokta nesnesi oluşturur.

- **Setter Fonksiyonları:**
  - **void setX(double xkordinat)**: x koordinatını ayarlar.
  - **void setY(double ykordinat)**: y koordinatını ayarlar.
  - **void set(double xkordinat, double ykordinat)**: Hem x hem de y koordinatlarını ayarlar.
- **Getter Fonksiyonları:**
  - **double getX() const**: x koordinatını döndürür.
  - **double getY() const**: y koordinatını döndürür.
- **Diğer Fonksiyonlar:**
  - **std::string toString() const**: Nokta nesnesini bir string olarak temsil eder.
  - **void yazdir() const**: Nokta nesnesini ekrana yazdırır.

  Bu sınıf, 2D koordinat sistemine ait noktaları temsil etmek için kullanılabilir.

```
DogruParcasi::DogruParcasi(Nokta dogrukordinatıortası, double dogruzunlugu, double
egim) {
    setortaNokta(dogrukordinatıortası);
    setUzunluk(dogruzunlugu);

    double yariuzunluk = dogruzunlugu / 2.0;

    xy1.setX(ortaNokta().getX() - yariuzunluk * std::cos(std::atan(tan(egim))));
    xy1.setY(ortaNokta().getY() - yariuzunluk * std::sin(std::atan(tan(egim))));

    xy2.setX(ortaNokta().getX() + yariuzunluk * std::cos(std::atan(tan(egim))));
    xy2.setY(ortaNokta().getY() + yariuzunluk * std::sin(std::atan(tan(egim))));
}

DogruParcasi::DogruParcasi(const DogruParcasi& Nesne) {
    xy1 = Nesne.getP1();
    xy2 = Nesne.getP2();
    xy_orta = Nokta{ (xy1.getX() + xy2.getX()) / 2, (xy1.getY() + xy2.getY()) / 2 };
    uzaklik = uzunluk();
}

DogruParcasi::DogruParcasi(Nokta xy11, Nokta xy22) {
    xy1 = xy11;
    xy2 = xy22;
    xy_orta = Nokta{ (xy1.getX() + xy2.getX()) / 2, (xy1.getY() + xy2.getY()) / 2 };
    uzaklik = uzunluk();
}
```

Bu C++ kodu, bir doğru parçasını temsil etmek için bir sınıfın implementasyonunu içerir. İşte her bir fonksiyonun görevlerinin açıklamaları:

1. **DogruParcasi (Nokta dogrukordinatıortası, double dogruzunlugu, double egim):**
    - Bu fonksiyon, bir doğru parçası oluşturmak için kullanılan parametreli kurucu fonksiyondur.
    - **dogrukordinatıortası** parametresi, doğru parçasının orta noktasını temsil eden bir **Nokta** nesnesidir.
    - **dogruzunlugu** parametresi, doğru parçasının toplam uzunluğunu temsil eder.
    - **egim** parametresi, doğru parçasının eğimini temsil eder.
    - **setortaNokta** fonksiyonu, doğru parçasının orta noktasını **dogrukordinatıortası** ile ayarlar.
    - **setUzunluk** fonksiyonu, doğru parçasının uzunluğunu **dogruzunlugu** ile ayarlar.
    - Ardından, doğru parçasının başlangıç ve bitiş noktaları olan **xy1** ve **xy2** noktaları, eğim ve orta nokta kullanılarak hesaplanır.
2. **DogruParcasi:**
    - Bu fonksiyon, bir **DogruParcasi** nesnesini kopyalamak için kullanılan kopyalama kurucu fonksiyondur.
    - **Nesne** parametresi, kopyalanacak **DogruParcasi** nesnesini temsil eder.
    - **getP1** ve **getP2** fonksiyonları, **Nesne**'nin başlangıç ve bitiş noktalarını alır ve bu noktaları sırasıyla **xy1** ve **xy2**'ye atar.
    - **xy_orta** noktası, **xy1** ve **xy2** noktalarının ortalaması olarak hesaplanır.
    - **uzunluk** fonksiyonu, doğru parçasının uzunluğunu hesaplar ve **uzaklik** üye değişkenine atanır.
3. **DogruParcasi::DogruParcasi(Nokta xy11, Nokta xy22):**
    - Bu fonksiyon, iki nokta parametresi alarak bir doğru parçası oluşturmak için kullanılan parametreli kurucu fonksiyondur.
    - **xy11** ve **xy22** parametreleri, doğru parçasının başlangıç ve bitiş noktalarını temsil eder.
    - **xy1** ve **xy2** noktaları, bu başlangıç ve bitiş noktalarına atanır.
    - **xy_orta** noktası, **xy1** ve **xy2** noktalarının ortalaması olarak hesaplanır.
    - **uzunluk** fonksiyonu, doğru parçasının uzunluğunu hesaplar ve **uzaklik** üye değişkenine atanır.

```
Nokta DogruParcasi::getP1() const {
    return xy1;
}

Nokta DogruParcasi::getP2() const {
    return xy2;
}

Nokta DogruParcasi::ortaNokta() const {
    return xy_orta;
}

void DogruParcasi::setP1(Nokta xy11) {
    xy1 = xy11;
```

```
}

void DogruParcasi::setP2(Nokta xy22) {
    xy2 = xy22;
}

void DogruParcasi::setortaNokta(Nokta x_orta1) {
    xy_orta = x_orta1;
}

void DogruParcasi::setUzunluk(double uzunluk) {
    uzaklik = uzunluk;
}

double DogruParcasi::uzunluk() const {
    double uzaklik = sqrt(pow(xy1.getX() - xy2.getX(), 2) + pow(xy1.getY() -
xy2.getY(), 2));
    return uzaklik;
}

std::string DogruParcasi::toString() const {
    yazdir();
    return    "x1y1: " + std::to_string(getP1().getX()) + " " +
std::to_string(getP1().getY()) + "\n" + "" + "x2y2: " +
std::to_string(getP2().getX()) + " " + std::to_string(getP2().getY()) + "\n";
}

void DogruParcasi::yazdir() const {
    std::cout << "x1y1: " << getP1().getX() << " " << getP1().getY() << std::endl;
    std::cout << "x2y2: " << getP2().getX() << " " << getP2().getY() << std::endl;
}

Nokta DogruParcasi::KesisimNoktasi(const Nokta& verilenNokta) const {
    // Doğru parçasının vektörünü hesapla
    double dogruParcaX = xy2.getX() - xy1.getX();
    double dogruParcaY = xy2.getY() - xy1.getY();

    // Doğru parçasının birim vektörünü hesapla
    double dogruParcaUzunluk = sqrt(dogruParcaX * dogruParcaX + dogruParcaY *
dogruParcaY);
    double birimDogruParcaX = dogruParcaX / dogruParcaUzunluk;
    double birimDogruParcaY = dogruParcaY / dogruParcaUzunluk;

    // Verilen noktanın doğru parçasına dik olan vektörünü hesapla
    double dikVektorX = verilenNokta.getX() - xy1.getX();
    double dikVektorY = verilenNokta.getY() - xy1.getY();

    // Verilen noktanın doğru parçasına dik olan uzaklığı hesapla
    double uzaklik = dikVektorX * birimDogruParcaX + dikVektorY * birimDogruParcaY;

    // Doğru parçasının üzerindeki kesişim noktasını hesapla
    double kesisimNoktasiX = xy1.getX() + uzaklik * birimDogruParcaX;
    double kesisimNoktasiY = xy1.getY() + uzaklik * birimDogruParcaY;

    // Yeni Nokta nesnesini oluştur ve döndür
    return Nokta(kesisimNoktasiX, kesisimNoktasiY);
}
```

Bu C++ kodu, bir doğru parçasını temsil etmek üzere tasarlanmış **DogruParcasi** sınıfının üye fonksiyonlarını içerir. Aşağıda her bir fonksiyonun açıklaması bulunmaktadır:

1. **DogruParcasi:getP1() const:**
   - Bu fonksiyon, doğru parçasının birinci noktasını (**xy1**) döndürür.
2. **DogruParcasi:getP2() const:**
   - Bu fonksiyon, doğru parçasının ikinci noktasını (**xy2**) döndürür.
3. **DogruParcasi:ortaNokta() const:**
   - Bu fonksiyon, doğru parçasının orta noktasını (**xy_orta**) döndürür.
4. **DogruParcasi:setP1(Nokta xy11):**
   - Bu fonksiyon, doğru parçasının birinci noktasını (**xy1**) belirtilen **xy11** noktasıyla ayarlar.
5. **DogruParcasi:setP2(Nokta xy22):**
   - Bu fonksiyon, doğru parçasının ikinci noktasını (**xy2**) belirtilen **xy22** noktasıyla ayarlar.
6. **DogruParcasi:setortaNokta(Nokta x_orta1):**
   - Bu fonksiyon, doğru parçasının orta noktasını (**xy_orta**) belirtilen **x_orta1** noktasıyla ayarlar.
7. **DogruParcasi:setUzunluk(double uzunluk):**
   - Bu fonksiyon, doğru parçasının uzunluğunu belirtilen **uzunluk** değeriyle ayarlar.
8. **DogruParcasi:uzunluk() const:**
   - Bu fonksiyon, doğru parçasının uzunluğunu hesaplar ve geri döndürür.
9. **DogruParcasi: toString() const:**
   - Bu fonksiyon, doğru parçasının bilgilerini bir string olarak formatlayarak döndürür. **yazdir()** fonksiyonunu çağırır.
10. **DogruParcasi:yazdir() const:**
    - Bu fonksiyon, doğru parçasının başlangıç ve bitiş noktalarını ekrana yazdırır.
11. **DogruParcasi:KesisimNoktasi(const Nokta& verilenNokta) const:**
    - Bu fonksiyon, doğru parçası ile verilen noktanın kesişim noktasını hesaplar ve bu noktayı bir **Nokta** nesnesi olarak döndürür. Kesişim noktası, doğru parçasının üzerindeki en yakın noktadır.

```
Daire::Daire(Nokta merkezxy1, double yarıcap1) {
    merkezxy = merkezxy1;
    yarıcap = yarıcap1;
}

Daire::Daire(const Daire& Nesne) {
    merkezxy = Nesne.merkezxy;
    yarıcap = Nesne.yarıcap;
}

Daire::Daire(const Daire& Nesne, double carpılacak_x_degeri) {
    merkezxy = Nesne.merkezxy;
    yarıcap = Nesne.yarıcap * carpılacak_x_degeri;
}
```

```
double Daire::alan() {
    double Alan = 3.14 * pow(getYarıcap(), 2);
    return Alan;
}

double Daire::cevre() {
    double cevre = 2 * 3.14 * yarıcap;
    return cevre;
}

std::string Daire::toString() {
    return std::to_string(merkezxy.getX()) + "," + std::to_string(merkezxy.getY()) +
"    " + std::to_string(yarıcap);
}

void Daire::setMerkezX(double x) {
    merkezxy.setX(x);
}

void Daire::setMerkezY(double y) {
    merkezxy.setY(y);
}

double Daire::getMerkezX() {
    return merkezxy.getX();
}

double Daire::getMerkezY() {
    return merkezxy.getY();
}

void Daire::yazdir() {
    std::cout << std::endl;
    std::cout << "merkez koordinatları " << "x: " << merkezxy.getX() << " y: " <<
merkezxy.getY() << std::endl;
    std::cout << "yarıcap: " << getYarıcap() << std::endl;
    std::cout << "Alan: " << alan() << std::endl;
    std::cout << "cevre: " << cevre() << std::endl;
}

double Daire::getYarıcap() {
    return yarıcap;
}

void Daire::setYarıcap(double yarıcap1) {
    yarıcap = yarıcap1;
}

int Daire::kesisim(const Daire& KesisimNesnesi) {
    // İki çemberin tamamen örtüşüp örtüşmediğini kontrol et
    double distance = std::sqrt(std::pow(KesisimNesnesi.merkezxy.getX() -
merkezxy.getX(), 2) + std::pow(KesisimNesnesi.merkezxy.getY() - merkezxy.getY(), 2));

    if ((distance == 0) && (getYarıcap() == KesisimNesnesi.yarıcap)) {
        return 1;
    }
```

```
    // İki çemberin kesişip kesişmediğini kontrol et
    distance = std::sqrt(std::pow(KesisimNesnesi.merkezxy.getX() - merkezxy.getX(),
2) + std::pow(KesisimNesnesi.merkezxy.getY() - merkezxy.getY(), 2));

    if (distance < (KesisimNesnesi.yarıcap + getYarıcap())) {
        return 0;
    }

    distance = std::sqrt(std::pow(getMerkezX() - KesisimNesnesi.merkezxy.getX(), 2) +
std::pow(getMerkezY() - KesisimNesnesi.merkezxy.getY(), 2));

    if ((distance + getYarıcap()) < KesisimNesnesi.yarıcap) {
        return 2;
    }
}
```

Bu C++ programı, bir "Daire" sınıfını tanımlar ve bu sınıfın bazı temel özelliklerini ve fonksiyonlarını içerir. İşte kodun genel açıklaması:

1. **Kurucu Fonksiyonlar:**
    - **Daire** sınıfının birinci kurucu fonksiyonu, bir dairenin merkez koordinatlarını (**Nokta** tipinden bir nesne) ve yarıçapını alarak bir daire nesnesi oluşturur.
    - İkinci kurucu fonksiyon, bir başka daire nesnesinin özelliklerini (merkez koordinatlarını ve yarıçapını) kopyalayarak yeni bir daire nesnesi oluşturur.
    - Üçüncü kurucu fonksiyon, bir başka daire nesnesinin özelliklerini kopyalayarak yarıçapını belirtilen bir değerle çarparak yeni bir daire nesnesi oluşturur.
2. **Alan ve Çevre Hesaplama:**
    - **alan** fonksiyonu, dairenin alanını hesaplar.
    - **cevre** fonksiyonu, dairenin çevresini hesaplar.
3. **Nokta İşlemleri ve Dönüşümler:**
    - **setMerkezX** ve **setMerkezY**, dairenin merkez koordinatlarını günceller.
    - **getMerkezX** ve **getMerkezY**, dairenin merkez koordinatlarını getirir.
4. **Daire Bilgilerini String Olarak Döndürme :**
    - **toString** fonksiyonu, dairenin merkez koordinatlarını ve yarıçapını bir string olarak döndürür.
5. **Yarıçap İşlemleri:**
    - **getYarıcap** fonksiyonu, dairenin yarıçapını getirir.
    - **setYarıcap** fonksiyonu, dairenin yarıçapını günceller.
6. **Daire Bilgilerini Ekrana Yazdırma:**
    - **yazdir** fonksiyonu, dairenin merkez koordinatları, yarıçapı, alanı ve çevresini ekrana yazdırır.
7. **İki Dairenin Kesişim Kontrolü:**
    - **kesisim** fonksiyonu, iki dairenin kesişim durumunu kontrol eder ve sonuca göre bir değer döndürür.
    - 1: İki daire tamamen örtüşüyorsa.

- 0: İki daire kesişiyorsa.
- 2: İki daire ayrılmışsa.

Bu kod, basit bir daire sınıfını temsil eder ve dairelerin özelliklerini yöneten temel fonksiyonları içerir.

```
Ucgen::Ucgen(Nokta xy11, Nokta xy22, Nokta xy33) {
    xy1 = xy11;
    xy2 = xy22;
    xy3 = xy33;
}

Nokta Ucgen::getXY1() {
    return xy1;
}

Nokta Ucgen::getXY2() {
    return xy2;
}

Nokta Ucgen::getXY3() {
    return xy3;
}

void Ucgen::setXY1(Nokta xy11) {
    xy1 = xy11;
}

void Ucgen::setXY2(Nokta xy22) {
    xy2 = xy22;
}

void Ucgen::setXY3(Nokta xy33) {
    xy3 = xy33;
}

std::string Ucgen::toString() {
    std::string result;
    result += xy1.toString() + "\n";
    result += xy2.toString() + "\n";
    result += xy3.toString() + "\n";
    return result;
}

double Ucgen::alan() {
    double uzunluk1, uzunluk2, uzunluk3, uzunlukort;
    DogruParcasi kenar1{ xy1,xy2 };
    DogruParcasi kenar2{ xy2,xy3 };
    DogruParcasi kenar3{ xy3,xy1 };
    uzunluk1 = kenar1.uzunluk();
    uzunluk2 = kenar2.uzunluk();
    uzunluk3 = kenar3.uzunluk();
    uzunlukort = (uzunluk1 + uzunluk2 + uzunluk3) / 2;
    return std::sqrt(uzunlukort * (uzunlukort - uzunluk1) * (uzunlukort - uzunluk2) *
(uzunlukort - uzunluk3));
}

double Ucgen::cevre() {
    DogruParcasi dogru1{ xy1,xy2 };
```

```
    DogruParcasi dogru2{ xy1,xy3 };
    DogruParcasi dogru3{ xy2,xy3 };
    return dogru1.uzunluk() + dogru2.uzunluk() + dogru3.uzunluk();
}

double* Ucgen::acilar() {
    double* dizi = new double[3];
    double acı1, acı2, acı3;
    DogruParcasi kenar1{ xy1,xy2 };
    DogruParcasi kenar2{ xy1,xy3 };
    DogruParcasi kenar3{ xy2,xy3 };
    acı1 = (acos((pow(kenar2.uzunluk(), 2) - pow(kenar1.uzunluk(), 2) -
pow(kenar3.uzunluk(), 2)) / (-2 * kenar1.uzunluk() * kenar3.uzunluk())) / 3.14159) *
180;
    acı2 = (acos((pow(kenar1.uzunluk(), 2) - pow(kenar2.uzunluk(), 2) -
pow(kenar3.uzunluk(), 2)) / (-2 * kenar2.uzunluk() * kenar3.uzunluk())) / 3.14159) *
180;
    acı3 = (acos((pow(kenar3.uzunluk(), 2) - pow(kenar2.uzunluk(), 2) -
pow(kenar1.uzunluk(), 2)) / (-2 * kenar1.uzunluk() * kenar2.uzunluk())) / 3.14159) *
180;
    dizi[0] = acı1;
    dizi[1] = acı2;
    dizi[2] = acı3;
    return dizi;
}
```

Bu C++ programı, bir "Ucgen" (üçgen) sınıfını tanımlar ve bu sınıfın özelliklerini yöneten fonksiyonları içerir. İşte kodun genel açıklaması:

1. **Kurucu Fonksiyon (Ucgen):**
    - **Ucgen** sınıfının kurucu fonksiyonu, üç noktanın koordinatlarını alarak bir üçgen nesnesi oluşturur.
2. **Getter ve Setter Fonksiyonlar:**
    - **getXY1**, **getXY2**, ve **getXY3** fonksiyonları, sırasıyla üçgenin birinci, ikinci ve üçüncü noktalarını getirir.
    - **setXY1**, **setXY2**, ve **setXY3** fonksiyonları, sırasıyla üçgenin birinci, ikinci ve üçüncü noktalarını günceller.
3. **Üçgen Bilgilerini String Olarak Döndürme:**
    - **toString** fonksiyonu, üçgenin her bir noktasının koordinatlarını bir string olarak döndürür.
4. **Üçgen Alanını Hesaplama :**
    - **alan** fonksiyonu, üçgenin alanını hesaplamak için üç kenarın uzunluklarını kullanır. Alan hesaplamasında üçgenin yarı çevresi ve Heron's formülü kullanılmıştır.
5. **Üçgen Çevresini Hesaplama:**
    - **cevre** fonksiyonu, üçgenin çevresini hesaplamak için üç kenarın uzunluklarını kullanır.
6. **Üçgen Açılarını Hesaplama:**
    - **acilar** fonksiyonu, üçgenin iç açılarını hesaplamak için kosinüs teoremi kullanır. Sonuçlar, dinamik olarak oluşturulan bir double dizisine atanır ve bu dizi döndürülür.

Bu kod, üçgenlerin temel özelliklerini hesaplamak ve yönetmek için bir sınıf oluşturan basit bir

programı içerir.

KAYNAKÇA:

https://www.w3schools.com/cpp/cpp_constructors.asp

https://www.w3schools.com/cpp/cpp_class_methods.asp