Shapes – Programozás alapjai 2. NHF

Dokumentáció

Baranyai Ferenc – LITIUP

Specifikáció

A megírt C++ kód 2 dimenziós szabályos alakzatok kezelésére alkalmas. Ezek a következőek:

- Háromszög
- Négyzet
- Kör

Az alakzatoknak van egy absztrakt osztálya (**Shape**) melyből származtatjuk a 3 darab gyermekosztályt (**Triangle, Square, Circle**). A szülő osztályban van eltárolva a középpontjuk és egy csúcsuk relatívan a középponthoz mivel így mozgatás esetén nem kell módosítani a csúcs pozícióját. A gyermekosztályokban nem tárolunk több csúcsot. Azokat a **GetVertices(int&)** virtuális függvény segítségével lehet lekérdezni. Emiatt nem kell minden gyermeknek külön megírni a **Move(const Vector&), Rotate(const double)** és **Scale(const double)** függvényeket sem, melyekkel mozgatni, forgatni és méretezni lehet. A leszármazottak még örökölik az **insideCircle(double)** és **pointInside(const Vector&)** virtuális függvényeket. Az előbbi megmondja, hogy egy origó körüli egységkörben benne van-e az adott alakzat, míg az utóbbi azt, hogy a paraméterként kapott pont benne van-e a síkidomban.

A **Paper** osztály egy heterogén tárolót valósít meg amiben a **Shape** osztály leszármazottjait lehet eltárolni. Ebben a tárolóban különböző módokon lehet az alakzatokat kezelni. Ilyen például a fájlból betöltés, illetve fájlba mentés. Ezek mellett lehet köztük keresni az origótól való távolság alapján a vagy hogy melyik tartalmaz egy adott pontot.

Készültek még a **Vector** és **mString** osztályok.

A **Vector** osztály gondoskodik a 2 dimenziós Vektorok kezeléséről és tárolásáról, így egyszerűbbé téve az alakzatok leírását és kezelését.

Az **mString** osztály stringek kezelését teszi lehetővé, ezzel egyszerűsítve a szabványos bemenetről olvasást és írást, illetve a menü vezérelt felhasználói interfészt.

A felhasználói felületet a **Menu** névtér segítségével valósítja meg a program. Itt több menüpontból lehet választani a kívánt feladat végrehajtásához. Ezek a következőek:

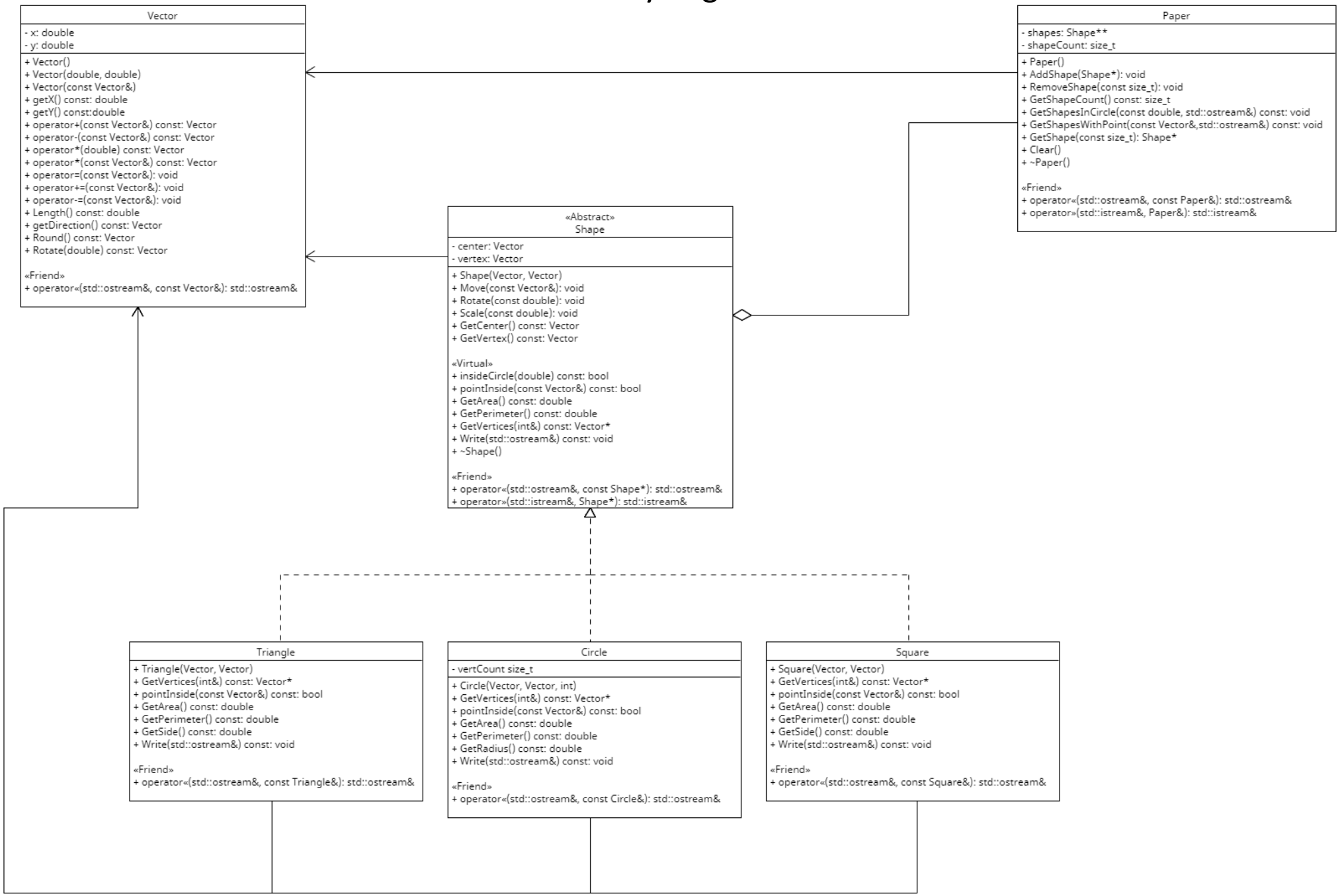
1. **Add shapes from file** – Alakzat hozzáadása fájlból
2. **Add shape from console** – Alakzat hozzáadása konzolról
3. **Remove shape** – Alakzat eltávolítása
4. **Move shape** – Alakzat elmozdítása
5. **Rotate shape** – Alakzat elforgatása
6. **Scale shape** – Alakzat méretezése
7. **Get shape in circle** – Alakzatok kiírása egy adott sugarú, origó körüli egységkörben
8. **Get shape with point** – Alakzatok kiírása, amelyek tartalmazzák az adott pontot
9. **List shapes** – Alakzatok kilistázása
10. **Save to file** – Alakzatok elmentése fájlba

Az alakzatokat a következő formátumban vannak elmentve és így is kell őket megadni, ha konzolról olvassuk be:

```
<típus> <kx> <ky> <cx> <cy>
```

ahol a **típus** a síkidom típusa mely lehet **Triangle** -> háromszög, **Circle** -> kör és **Square** -> négyzet. Emellett **kx** és **ky** az alakzat középpontjának x és y koordinátái és a **cx** és **cy** az egyik csúcsának az x és y koordinátái.

Osztálydiagram



Osztályok leírása

Vector osztály

2 dimenziós vektorok és koordináták tárolására alkalmas adatszerkezet. Támogatja a főbb vektorokkal elvégezhető műveleteket.

vector.h fájlban van definiálva

Privát adattagok

double x – x koordináta
double y – y koordináta

Publikus tagfüggvények

Konstruktorok

Vector()
Vector(double xt, double yt)
Vector(const Vector& v)

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Vector v)

Operátor függvények

Vector operator+(const Vector& v) const
Vector operator-(const Vector& v) const
Vector operator*(double d) const
Vector operator*(const Vector& v) const
void operator=(const Vector& v)
void operator+=(const Vector& v)
void operator-=(const Vector& v)

Getter függvények

double getX()
double getY()

Egyéb függvények

double Length() const
Vector getDirection() const
Vector Round() const
Vector Rotate(double deg) const

Részletes dokumentáció a Vector osztály tagfüggvényeiről

Konstruktorok

Vector()

Paraméter nélküli konstruktor. Az x és y koordinátákat 0-ra állítja be.

Vector(double xt, double yt)

Paraméteres konstruktor, mellyel megadható kezdőérték a vektoroknak.

Paraméterei

double xt - Vektor x koordinátája

double yt - Vektor y koordinátája

Vector(const Vector& v)

Másoló konstruktor, mellyel egy másik vektor másolható le.

Paraméterei

const Vector& v - Másolandó vektor

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Vector& v)

Vektor kiírása szabványos kimenetre.

Paraméterei

std::ostream& v – Szabványos kimenet helye

const Vector& os - Kiírandó vektor

Operátor függvények

Vector operator+(const Vector& v) const

Összeadás operátor. Két vektor összeadását teszi lehetővé.

Paraméterek

const Vector& v - Vektor, amit hozzá akarunk adni az aktuális vektorhoz

Visszatérési érték

Aktuális vektor és a paraméter vektor összege.

Vector operator-(const Vector& v) const

Kivonás operátor. Két vektor kivonását teszi lehetővé.

Paraméterek

const Vector& v - Vektor, amit ki akarunk vonni az aktuális vektorból

Visszatérési érték

Aktuális vektor és a paraméter vektor különbsége.

Vector operator*(double d) const

Szorzás operátor. Vektor szorzása skalárral.

Paraméterek

const Vector& d - Szorzó

Visszatérési érték

Aktuális vektor és a paraméter érték szorzata

Vector operator*(const Vector& v) const

Szorzás operátor. Két vektor koordinánkénti összeszorzása.

Paraméterek

const Vector& v - Vektor, amivel megszorozzuk az aktuális vektort

Visszatérési érték

Aktuális vektor és a paraméter vektor koordinánkénti szorzata

void operator=(const Vector& v)

Értékadó operátor.

Paraméterek

const Vector& v - Vektor, aminek az értékét át akarjuk venni

void operator+=(const Vector& v)

Plusz-egyenlő operátor. Az aktuális vektorhoz hozzáadja a paramétert. Ezzel azt felülírja.

Paraméterek

const Vector& v - Vektor, amit hozzá akarunk adni az aktuális vektorhoz

void operator-=(const Vector& v)

Mínusz-egyenlő operátor. Aktuális vektorból kivonja a paraméter vektort. Ezzel azt felülírja.

Paraméterek

const Vector& v - Vektor, amit ki akarunk vonni az aktuális vektorból

Getter függvények

double getX() const

Visszatérési érték

Vektor x koordinátája

double getY() const

Visszatérési érték

Vektor y koordinátája

Egyéb függvények

double Length() const

Visszatérési érték

Vektor hossza

Vector getDirection() const

Visszatérési érték

Vektor 1 hosszúságú irányvektorát

Vector Round() const

Visszatérési érték

Vektor kerekített koordinátákkal

Vector Rotate(double deg) const

Elforgatja a vektort

Paraméterek

double deg – Az elforgatás nagysága fokban megadva

Visszatérési érték

Elforgatott vektor

mString osztály

Szöveg tárolására és szerkesztésére alkalmas adatszerkezet.

mystring.h fájlban van definiálva

Privát adattagok

char* text – Szöveget tartalmazó karaktertömb
size_t len – Szöveg hossza a záró ’\0’ karaktert nem beleszámolva

Publikus tagfüggvények

Konstruktorok/Destruktorok

mString()
mString(const char* t)
mString(const mString& str)
~mString()

Barát függvények

friend ostream& operator<<(ostream& os, const mString& str)
friend istream& operator>> (istream& is, mString& str)

Operátor függvények

void operator=(const mString& str)
void operator=(const char* str)
mString operator+(const mString& str) const
mString operator+(const char* str) const
mString operator+(const char ch) const
void operator+=(const mString& str)
void operator+=(const char ch)
void operator+=(const char* str)
bool operator==(const mString& str) const
bool operator==(const char* str) const
char operator[](size_t index) const
char& operator[](size_t index)

Getterek

char* getText() const

Egyéb függvények

int Length() const
mString toLower() const
mString toUpper() const
int Split(char sep,mString*& out) const
mString RemoveAt(size_t startIndex, size_t endIndex) const
void clear()

Részletes dokumentáció a mString osztály tagfüggvényeiről

Konstruktorok/Destruktorok

mString()

Alapértelmezett konstruktor, ami létrehoz egy üres karaktertömböt és beállítja a hosszát 0-ra.

mString(const char* t)

Paraméteres konstruktor. A megadott karaktertömböt bemásolja a saját tömbjébe.

Paraméterek

const char* t – Karaktertömb, amit az osztály át fog venni

mString(const mString& str)

Másoló konstruktor. Lemásolja a paraméterét.

Paraméterek

const mString& str – A lemásolandó mString osztály

~mString()

Destruktor, ami törli a lefoglalt memóriahelyeket.

Barát függvények

friend ostream& operator<<(ostream& os, const mString& str)

mString osztály kiírása szabványos kimenetre

Paraméterek

ostream& os – Szabványos kimenet helye

const mString& str – Kiírandó mString

friend istream& operator>> (istream& is, mString& str)

mString osztály beolvasása szabványos bemenetről

Paraméterek

istream& is – Szabványos bemenet helye

mString& str – mString amibe a beolvasás történik

Operátor függvények

void operator=(const mString& str)

Értékadó operátor mellyel mString típusú értéket lehet odaadni az mString osztálynak

Paraméterek

const mString& str – Átadandó mString osztály

void operator=(const char* str)

Értékadó operátor mellyel karaktertömb típusú értéket lehet odaadni az mString osztálynak

Paraméter

const char* str – Odaadandó tömbre mutató pointer

mString operator+(const mString& str) const

Összeadó operátor mely 2 darab mString osztály szövegét tudja egybevonni

Paraméterek

const mString& str – Balérték amit az mString szöveg végére szúr be

Visszatérési érték

Összevont mString osztály

mString operator+(const char* str) const

Összeadó operátor mellyel mString osztályt lehet összevonni karaktertömbbel

Paraméterek

const char* str – Karaktertömbre mutató pointert melyet, az mString szöveg végére szúr be

Visszatérési érték

Karaktertömbbel összevont mString osztály

mString operator+(const char ch) const

Összeadó operátor mellyel a szöveg végére be lehet szúrni egy karakter

Paraméterek

const char ch – beszúrandó karakter

Visszatérési érték

mString aminek a végére már be van szúrva a karakter

void operator+=(const mString& str)

Plusz-egyenlő operátor mely mString végére szúr egy másik mString-et

Paraméterek

const mString& str – A beszúrandó mString

void operator+=(const char ch)

Plusz-egyenlő operátor mely mString végére szúr be egy karaktert

Paraméterek

const char ch – Beszúrandó karakter

void operator+=(const char* str)

Plusz-egyenlő operátor mely mString végére szúr be egy karaktertömböt

Paraméterek

const char* str – Beszúrandó karaktertömbre mutató pointer

bool operator==(const mString& str) const

Egyenlőség ellenőrző operátor mely két mString szövegét hasonlítja össze és megmondja, hogy megegyeznek-e.

Paraméterek

const mString& str – mString amivel összehasonlítunk

Visszatérési érték

Igaz értékkel ér vissza, ha a két mString szövege megegyezik

bool operator==(const char* str) const

Összehasonlító operátor, mely mString -et hasonlít össze egy karaktertömbbel.

Paraméterek

const char* str – Összehasonlítandó karaktertömbre mutató pointer

Visszatérési érték

Igaz, ha a két szöveg megegyezik

char operator[](size_t index) const

Konstans indexelő operátor mellyel a szövegben lévő karakterekre lehet ránézni, de azt nem lehet módosítani.

Paraméter

size_t index – Index melyen az elemet vissza szeretnénk kapni

Visszatérési érték

Konstans karakter az indexedik helyről

char& operator[](size_t index)

Indexelő operátor mellyel a szövegben lévő karakterekre lehet ránézni és akár módosítani is

Paraméter

size_t index – Index melyen az elemet vissza szeretnénk kapni

Visszatérési érték

Karakter referencia az indexedik helyről

Getterek

char* getText() const

Konstans függvény mellyel meg lehet nézni az mString-ben lévő szöveget viszont azt tmódosítani nem lehet

Visszatérési érték

Szöveget tartalmazó tömb vektora

Egyéb függvények

int Length() const

Szöveg hosszának lekérdezése

Visszatérési érték

Szöveg hossza

mString toLower() const

Szöveg kisbetűssé alakítása

Visszatérési érték

Eredeti szöveget kisbetűsen tartalmazó mString

mString toUpper() const

Szöveg nagybetűssé alakítása

Visszatérési érték

Eredeti szöveget nagybetűsen tartalmazó mString

int Split(char sep,mString*& out) const

Szöveg feldarabolása karakterek mentén

Paraméterek

char sep – Karakter, ami szerint feldaraboljuk a szöveget

mString*& out – mString tömb pointere melybe a feldarabolt szöveg kerül

Visszatérési érték

Szövegdarabok száma

mString RemoveAt(size_t startIndex, size_t endIndex) const

Szövegből való kimetszés két index között

Paraméterek

size_t startIndex – A kimetszés kezdetének helye (indexedik hely is kimetszésre kerül)

size_t endIndex – A kimetszés végének helye (indexedik hely is kimetszésre kerül)

Visszatérési érték

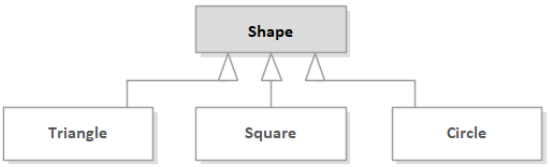
Kimetszett rész mentes szöveg

void clear()

Szöveg kiürítése a tárolóból

Shape osztály

Absztrakt osztály melyből 3 különböző szabályos alakzat osztálya származik. Az alakzatokat egy középpont és az egyik csúcsukkal lehet megadni. A csúcs koordinátái relatívan a középponthoz vannak megadva, ezzel egyszerűsítve az alakzatok transzformációit.



Az osztály 3 darab leszármazottja a **Triangle**, ami szabályos háromszög kezelését valósítja meg. A **Square** a szabályos négyszöget valósítja meg és a **Circle** egy kör kezelését teszi lehetővé.

shape.h fájlban van definiálva

Privát adattagok

Vector center – Az alakzat közepe
Vector vertex – Az alakzat egy pontja

Publikus tagfüggvények

Konstruktorok/Destruktorok

Shape(Vector c, Vector v)
virtual ~Shape()

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Shape* shape)
friend std::istream& operator>>(std::istream& input, Shape* shape)

Virtuális függvények

virtual bool pointInside(const Vector& p) const = 0
virtual bool insideCircle(double rad) const
virtual double GetArea() const = 0
virtual double GetPerimeter() const = 0
virtual Vector* GetVertices(int& n) const = 0
virtual void Write(std::ostream& os) const = 0

Getterek

Vector GetCenter() const
Vector GetVertex() const

Egyéb függvények

void Move(const Vector& v)
void Rotate(const double alpha)
void Scale(const double scale)

Részletes dokumentáció a Shape osztály tagfüggvényeiről

Konstruktorok/Destruktorok

Shape(Vector c, Vector v)

Shape osztály paraméteres konstruktora, mely létrehozza az alakzatot a középpont és az egyik pontjának megadásával

Paraméterek

Vector c – Az alakzat középpontja

Vector v – Az alakzat egyik csúcsa

virtual ~Shape()

Virtuális destruktor, melyet a gyermekosztályok megörökölhetnek

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Shape* shape)

Alakzatok kiírása szabványos kimenetre szabványos formátumban (<név> <cx> <cy> <vx> <vy>).

Parméterek

std::ostream& os – Szabványos kimenet helye

const Shape* shape – Kiírandó alakzat pointere

friend std::istream& operator>>(std::istream& input, Shape* shape)

Alakzatok beolvasása szabványos bemenetről

Paraméterek

sdt::istream& input – Szabványos bemenet helye

Shape* shape – Alakzat pointer amibe beolvassunk

Virtuális függvények

virtual bool pointInside(const Vector& p) const = 0

Megállapítja, hogy egy adott pont az alakzaton belül található-e.

Paraméterek

const Vector& p – Pont, amit ellenőrzünk, hogy benne van-e az alakzatban

Visszatérési érték

Logikai igaz, ha a pont az alakzaton belül van

virtual bool insideCircle(double rad) const

Megállapítja, hogy az alakzat minden csúcsa a rad sugarú, origó középpontú egységkörben van-e.

Paraméterek

double rad – Origó körüli egységkör sugara

Visszatérési érték

Logikai igaz, ha az alakzat minden csúcsa az egységkörön belül van

Öröklés

Nem tisztán virtuális függvény. A Triangle és Square osztály megörökli felülírás nélkül, viszont a Circle osztály felülírja, mivel egy körnek végtelen mennyiségű pontja lehet ezért ezt máshogy számolja ki.

virtual double GetArea() const = 0

Alakzat területének kiszámítása

Visszatérési érték

Az alakzat területe

virtual double GetPerimeter() const = 0

Alakzat kerületének kiszámolása

Visszatérési érték

Az alakzat kerülete

virtual Vector* GetVertices(int& n) const = 0

Az alakzat csúcsainak lekérése

Paraméterek

int& n – Ebbe fog belekerülni a csúcsok száma

Vissztérési érték

Egy n hosszú vektor tömbre mutató pointer, ami tartalmazza az alakzat csúcsait

virtual void Write(std::ostream& os) const = 0

Kiírja az alakzat nevét szabványos kimenetre, segít az egyéb kiírásoknál.

Paraméterek

std::ostream& os – Szabványos kimenet helye

Getterek

Vector GetCenter() const

Középpont lekérdezése

Visszatérési érték

Az alakzat középpontja

Vector GetVertex() const

Alapértelmezett csúcs lekérdezése

Visszatérési érték

Az alakzat egy csúcsa

Egyéb függvények

void Move(const Vector& v)

Az alakzat elmozgatása

Paraméterek

const Vector& v – Az elmozgatása vektora

void Rotate(const double alpha)

Alakzat elforgatása

Paraméterek

const double alpha – Elforgatás szöge fokban megadva

void Scale(const double scale)

Alakzat méretezése

Paraméterek

const double scale – Méretezés nagysága

Triangle osztály

A Shape absztrakt osztály leszármazottja. Szabályos háromszögek kezelésére alkalmas osztály

shape.h fájlban van definiálva

Publikus tagfüggvények

Konstruktorok

Triangle(Vector c, Vector v)

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Triangle& triangle)

Örökölt függvények

Vector* GetVertices(int& n) const
bool pointInside(const Vector& p) const
double GetArea() const
double GetPerimeter() const
void Write(std::ostream& os) const

Egyéb függvények

double GetSide() const

Részletes dokumentáció a Triangle osztály tagfüggvényeiről

Konstruktorok

Triangle(Vector c, Vector v)

Paraméteres konstruktor mely átadja a középpontot és az egyik csúcs koordinátáját a szülő osztálynak

Paraméterek

Vector c – Háromszög középpontja
Vector v – Háromszög egyik csúcsa

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Triangle& triangle)

Háromszög adatainak kiíratása szabványos kimenetre, mint például a neve, terület és kerület.

Paraméterek

std::ostream& os – Szabványos kimenet helye
const Triangle& triangle – Kiírandó háromszög

Egyéb függvények

double GetSide() const

Megadja a háromszög oldalának hosszát. Mivel szabályos háromszögeket tudunk csak létrehozni ezért minden oldal hossza egyenlő lesz.

Visszatérési érték

Háromszög oldalhossza

Square osztály

A Shape absztrakt osztály leszármazottja. Szabályos négyzetek kezelésére alkalmas osztály

shape.h fájlban van definiálva

Publikus tagfüggvények

Konstruktorok

Square(Vector c, Vector v)

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Square& square)

Örökölt függvények

Vector* GetVertices(int& n) const
bool pointInside(const Vector& p) const
double GetArea() const
double GetPerimeter() const
void Write(std::ostream& os) const

Egyéb függvények

double GetSide() const

Részletes dokumentáció a Square osztály tagfüggvényeiről

Konstruktorok

Square(Vector c, Vector v)

Paraméteres konstruktor mely átadja a középpontot és az egyik csúcsot a szülő osztálynak

Paraméterek

Vector c – Négyzet középpontja
Vector v – Négyzet egyik csúcsa

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Square& square)

Négyzet adatainak kiíratása szabványos kimenetre, mint például a neve, terület és kerület.

Paraméterek

std::ostream& os – Szabványos kimenet helye
const Square& square – Kiírandó négyzet

Egyéb függvények

double GetSide() const

Megadja a négyzet oldalának hosszát. Mivel szabályos négyzeteket tudunk csak létrehozni ezért minden oldal hossza egyenlő lesz.

Visszatérési érték

Négyzet oldalhossza

Cirlce osztály

A Shape absztrakt osztály leszármazottja. Kör alakzat kezelésére alkalmas osztály

shape.h fájlban van definiálva

Privát adattagok

size_t vertCount – a kör pontjainak száma, minél magasabb annál pontosabb

Publikus tagfüggvények

Konstruktorok

Circle(Vector c, Vector v, int n = 36)

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Circle& circle)

Örökölt függvények

Vector* GetVertices(int& n) const
bool pointInside(const Vector& p) const
double GetArea() const override
double GetPerimeter() const
void Write(std::ostream& os) const

Egyéb függvények

double GetRadius() const

Részletes dokumentáció a Circle osztály tagfüggvényeiről

Konstruktorok

Circle(Vector c, Vector v, int n = 36)

Paraméteres konstruktor, ami a kör középpontját és az egyik pontját odaadja az ősosztálynak és beállítja, hogy hány darab pontból álljon a kör.

Paraméterek

Vector c – Kör középpontja
Vector v – Kör egyik pontja
int n – Kör pontjainak száma, alapértelmezetten 36

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Circle& circle)

Kör adatainak kiíratása szabványos kimenetre, mint például a neve, terület és kerület.

Paraméterek

std::ostream& os – Szabványos kimenet helye
const Circle& circle – Kiírandó kör

Egyéb függvények

double GetRadius() const

Kör sugarának kiszámítása

Visszatérési érték

Kör sugara

Paper osztály

Shape osztály heterogén tároló adatszerkezete, mellyel alakzatok csoportos kezelése valósítható meg.

paper.h fájlban van definiálva

Privát adattagok

Shape** shapes – Az alakzat pointereket tartalmazó tömb
size_t shapeCount – A tömbben tárolt alakzatok száma

Publikus tagfüggvények

Konstruktorok/Destruktorok

Paper()
~Paper()

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Paper& paper)
friend std::istream& operator>>(std::istream& input, Paper& paper)

Egyéb függvények

void AddShape(Shape* shape)
void RemoveShape(const size_t idx)
size_t GetShapeCount() const
void GetShapesInCircle(const double r, std::ostream& os = std::cout) const
void GetShapesWithPoint(const Vector& v,std::ostream& os = std::cout) const
Shape* GetShape(const size_t idx)
void Clear()

Részletes dokumentáció a Paper osztály tagfüggvényeiről

Konstruktorok/Destruktorok

Paper()

Létrehoz egy üres shapes tömböt és lenullázza a shapeCount változót

~Paper()

Eltávolítja a Shape** típusú tároló tömböt.

Barát függvények

friend std::ostream& operator<<(std::ostream& os, const Paper& paper)

Paper típusú objektumban tárolt alakzatokat kiírja szabványos kimenetre a szabványos formátumban (<név> <cx> <cy> <vx> <vy>). Alkalmas az osztály elmentésére egy fájlban.

Paraméterek

std::ostream& os – Szabványos kimenet helye

const Paper& paper – Kiírandó objektum

friend std::istream& operator>>(std::istream& input, Paper& paper)

Paper típusú objektumba lehet vele beolvasni alakzatokat. Alkalmas elmentett Paper objektum mentésének betöltésére is.

Paraméterek

std::istream& input – Szabványos bemenet helye

const Paper& paper – Objektum amibe a beolvasás történik

Egyéb függvények

void AddShape(Shape* shape)

Alakzat hozzáadása a tárolóhoz

Paraméterek

Shape* shape – Hozzáadandó alakzat pointere

void RemoveShape(const size_t idx)

Alakzat eltávolítása a tárolóból

Paraméterek

const size_t idx – Eltávolítandó elem indexe

size_t GetShapeCount() const

Alakzatok számának lekérdezése

Visszatérési érték

Alakzatok száma

void GetShapesInCircle(const double r, std::ostream& os = std::cout) const

Megkeresi és kiírja szabványos kimenetre azokat az alakzatokat, amelyek r sugarú, origó körüli egységkörben találhatóak.

Paraméterek

const double r – Origó körüli egységkör sugara

std::ostream& os – Szabványos kimenet helye, alapértelmezetten std::cout-al a konzolra ír

void GetShapesWithPoint(const Vector& v,std::ostream& os = std::cout) const

Megkeresi és kiírja szabványos kimenetre azokat az alakzatokat, amelyek tartalmazzák a v pontot.

Paraméterek

const Vector& v – A pont melyet tartalmaznia kell az alakzatnak

std::ostream& os – Szabványos kimenet helye, alapértelmezetten std::cout-al a konzolra ír

Shape* GetShape(const size_t idx)

Alakzat lekérése index alapján

Paraméterek

const size_t idx – Alakzat indexe

Visszatérési érték

A lekérdezett alakzatra mutató pointer

void Clear()

Tároló kiürítése. Létrehoz utána egy új üres tárolót

namespace Menu

A menü vezérlést valósítja meg. Az elkészítése során fontos szempont volt a különböző be-és kimenetek közötti kompatibilitás ezért a fő függvény meghívásánál meg lehet adni, hogy honnan szeretnénk az adatokat beolvasni, hogy hova szeretnék a számítási adatokat kiírni és hogy hova szeretnénk tenni az interfész által létrehozott párbeszéd elemeket. Így ki lehet szűrni gépi ellenőrzésnél a felesleges párbeszéd elemeket és a fontos számítási adatokat.

menu.h fájlban van definiálva

Tagfüggvények

void DrawMain(int shapeCount, std::ostream& os)

Menü megjelenítése

Paraméterek

int shapeCount – Betöltött alakzatok száma

std::ostream& os - Szabványos kimenet helye

int GetIndex(Paper& paper, std::ostream& os, std::istream& input)

Index párbeszédpanel megjelenítése, ahol indexet lehet bekérni

Paraméter

Paper& paper – Betöltött alakzatokat tartalmazó objektum

std::ostream& os – Panel szabványos kimenetének helye

std::istream& input – Panel szabványos bemenetének helye

Visszatérési érték

Bekért index

Shape* Selector(Paper& paper, std::ostream& os, std::istream& input)

Elemkiválasztópanel megjelenítése, ahol a Paper tárolóból lehet kiválasztani egy elemet.

Paraméterek

Paper& paper – Betöltött alakzatokat tároló objektum

std::ostream& os – Panel szabványos kimenetének helye

std::istream& input – Panel szabványos bemenetének helye

Visszatérési érték

A kiválasztott elem

template<typename T>
T GetParameter(const mString& question, std::ostream& os, std::istream& input)

Egyéb paraméterek bekérésére alkalmas függvény. Sablon segítségével minden típusú érték bevitelét támogatja, ami rendelkezik inserter operátorral.

Paraméterek

const mString& question – Bekérő szöveg

std::ostream& os – Panel szabványos kimenetének helye

std::istream& input – Panel szabványos bemenet helye

Visszatérési érték

Bekért paraméter

void menu(std::istream& input, std::ostream& value_out, std::ostream& interface_out)

A névtér fő függvénye, ami kezeli annak menetét és a bevitt adatokat.

Paraméterek

std::istream& input – Menü szabványos bemenet helye

std::ostream& value_out – Menü adatainak szabványos kimenetének helye

std::ostream& os – Menü párbeszéd szabványos kimenetének helye

Tesztelés

A tesztelés a **shape_test.cpp** fájlban történik, ami két részből áll. Az első részben az osztályok tagfüggvényeinek tesztje fut le majd ezt követve a felhasználói interfészé. A teszteléshez a **gtest_lite.h** és a **memtrace.h** fájlokat használtam. A tesztelés során a következők lettek tesztelve:

- *mString osztály*
- *Vector osztály*
- *Shape osztály és leszármazottja*
- *Paper osztály*
- *Menu névtér*

A tesztelés során igyekeztem a legtöbb függvény minden funkcióját kipróbálni. A tesztek során a **memtrace** nem jelzett memória szivárgást vagy túlcímzést.

A **Menu** névtér tesztelése során a bemenetet egy fájlból olvasta ki a program, majd a fontos kimeneteket kiírta egy másik fájlba. A kimeneti fájlt majd újra megnyitotta a teszt program és ellenőrizte benne, hogy a kapott értékek helyesek-e.

A teszteléshez a következő segédfüggvények készültek:

template<typename T>
void ToFile(const T& obj, const mString& path)

Objektumok fájlba való kiírása. Sablon segítségével minden osztály kiírható, ami rendelkezik inserter operátorral.

Paraméterek

const T& obj – Kiírandó objektum

const mString& path – Fálj helye amibe a beírás történik

bool CheckOutput(const mString& path, const mString& expected)

Ellenőrzi a kimeneti fájlban lévő értékek helyességét.

Paraméterek

const mString& path – Kimeneti fájl helye

const mString& expected – Elvárt érték

Visszatérési érték

Logikai igaz, ha a fájl tartalma megegyezik az elvártal

double Round(double num, int decimals)

Double típusú számokat kerekíti fel, hogy össze lehessen őket hasonlítani.

Paraméterek

double num – Felkerekítendő szám

int decimals – Tizedesjegyek száma

Visszatérési érték

Felkerekített double típusú szám

Ezek a függvények a **shape_test.cpp** fájlban lettek létrehozva.