

Koordinátageometria

Feladat:

Rögzítsen a síkon egy pontot, és töltsön fel egy láncolt listát különféle szabályos (kör, szabályos háromszög, négyzet, szabályos hatszög) síkidomokkal! Keresse meg melyik síkidom van legközelebb a ponthoz! Közelségen kör esetén a körvonaltól vett távolságot, sokszög esetén a legközelebbi csúcstól mért távolságot értjük. Minden síkidom reprezentálható a középpontjával és az oldalhosszal, illetve a sugárral, ha feltesszük, hogy a sokszögek esetében az egyik oldal párhuzamos a koordináta rendszer vízszintes tengelyével, és a többi csúcs ezen oldalra fektetett egyenes felett helyezkedik el. A síkidomokat szövegfájlból töltsse be! A fájl első sorában szerepeljen a síkidomok száma, majd az egyes síkidomok. Az első jel azonosítja a síkidom fajtáját, amit követnek a középpont koordinátái és a szükséges hosszúság. A feladatokban a beolvasáson kívül a síkidomokat egységesen kezelje, ennek érdekében a síkidomokat leíró osztályokat egy közös ősosztályból származtassa!

Ismert adatok:

- Síkidomok száma;
- Síkidom oldalainak száma;
- Középpont koordinátái (x, y) ;
- Oldalhosszúság;

Kiszámítandó adat:

Egy adott $P(x, y)$ ponthoz legközelebbi síkidom meghatározása.

A számítási műveletek leegyszerűsítéséért feltesszük, hogy az egyes síkidomok egyik oldala párhuzamos a koordináta-rendszer vízszintes (x) tengelyével.

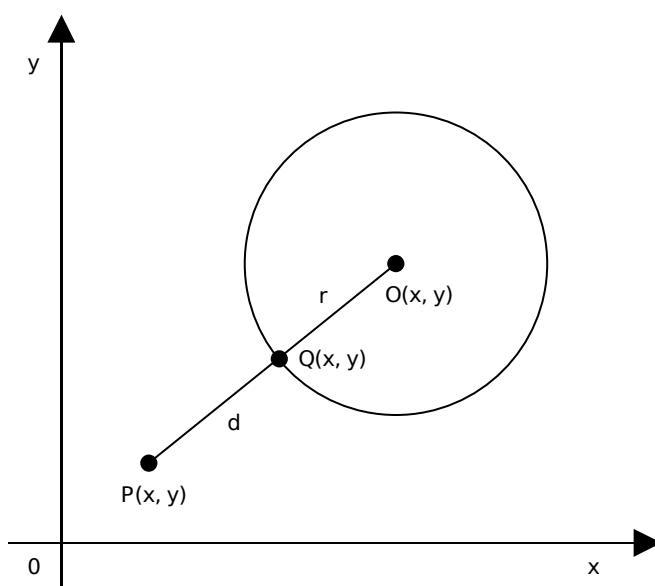
Megelőző számítások:

A megoldási folyamat megkönnyítéséhez most megadjuk a szükséges képleteket. A síkidomok reprezentációjában megadott paraméterektől függően a feladat ezekkel a képletekkel lesz megoldva.

A körrel kapcsolatos adatok ábrázolása az 1. ábrán látható.

A körvonal és a $P(x, y)$ pont közötti távolságot a „ d ” szakasz jelöli. Ez a távolság a $P(x, y)$ pont, $O(x, y)$, vagyis a kör középpontja, valamint a kör sugarának ismeretében könnyen meghatározható:

1. Először meg kell határozni $O(x, y)$ és $P(x, y)$ pontok távolságát. Ezt koordinátáik



ábra 1: Kör ábrázolása

ismeretében egy lépésben megtehetjük:

$$e = \sqrt{(O.x - P.x)^2 + (O.y - P.y)^2}$$

A képletben a *Pitagorasz-tételt* alkalmaztuk.

2. Tudjuk, hogy $e = d + r$. Nekünk a „d” változó értékét kell kifejezni, ezt pedig az egyenlet átrendezésével tehetjük meg:

$$d = e - r$$

3. Hogy a programnak ne kelljen még külön az „e” változót is kezelnie, helyettesítsük be az első egyenlet értékét a második egyenletbe! Ekkor a következő egyenletet kapjuk eredményül:

$$d = \sqrt{(O.x - P.x)^2 + (O.y - P.y)^2} - r$$

ahol a „d” változó a körvonal és a P(x, y) pont távolsága.

A körrel kapcsolatos képlet meghatározása után most rátérhetünk tetszőleges, n oldalú, szabályos síkidomokkal kapcsolatos számítási műveletek tárgyalására. A feladat szövegében definiált távolság-fogalommal élve, egy tetszőleges P(x, y) pont síkidomtól való távolságát a síkidom P(x, y) ponthoz legközelebbi csúcsának P(x, y) ponttól mért távolsága adja. A távolságot kiszámító képletet egy konkrét síkidomon (szabályos ötszögön) keresztül határozzuk meg.

A példában szereplő szabályos ötszög adatainak ábrázolása a 2. ábrán látható.

A P(x, y) pont és a síkidom (a példában szabályos ötszög) távolságát ezúttal is a „d” szakasz jelöli. A távolság meghatározása az előző esethez képest jóval több lépésben történik:

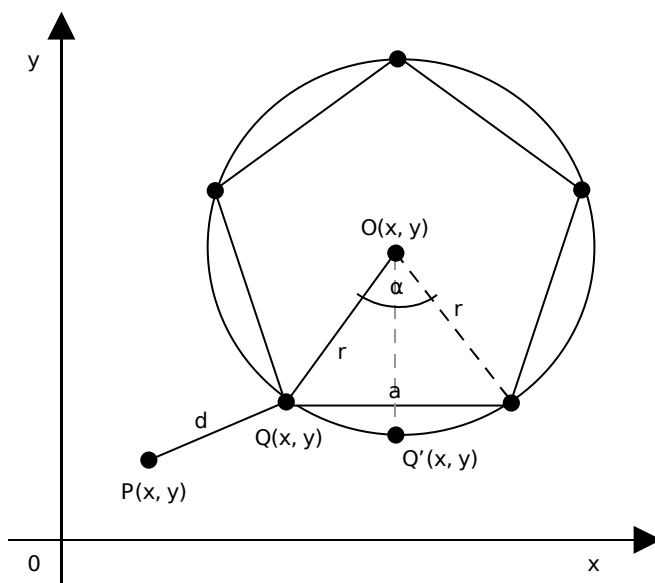
1. α szög meghatározása. Ezt egy lépésben megtehetjük: $\alpha = \frac{360}{n}$, ahol az „n” változó a megadott síkidom oldalainak száma (a példában: n=5);
2. A síkidom köré írt kör sugarának meghatározása; erre az adatra a síkidom egyes csúcsainak – elforgatással történő – meghatározásához lesz szükségünk.

Felhasználva szögfüggvényekkel kapcsolatos ismereteinket, a sugár megkapható a következő egyenletből:

$$\sin\left(\frac{\alpha}{2}\right) = \frac{\frac{a}{2}}{r}, \text{ egyszerűbben: } \sin\left(\frac{\alpha}{2}\right) = \frac{a}{2r}. \text{ Az egyenletet megfelelően átrendezve megkapjuk a sugár hosszát}$$

meghatározó képletet: $r = \frac{a}{\sin(\frac{\alpha}{2}) \cdot 2}$, ahol „a” a síkidom oldalának hossza, α pedig a síkidom középpontjából az „a” szakasz két végpontjába húzott szakaszok (vagyis a keresett sugarak) által bezárt szög.

3. A sugár hosszának ismeretében most már meg tudjuk határozni a 2. ábrán látható Q'(x, y) pontot, mely az elforgatás alapját fogja képezni: $Q'(x, y) = O(x, y - r)$, vagyis az O(x, y) pont „y” koordinátájából egyszerűen kivonjuk az „r” sugár hosszát, így megkapjuk a Q'(x, y) pontot.



ábra 2: Szabályos ötszög ábrázolása

4. Most meghatározzuk a síkidom első csúcsának koordinátáit – amit a 2. ábrán $Q(x, y)$ pont jelöl – majd az így kapott képletet általánosítjuk az összes többi pontra is.

Ahhoz, hogy megkapjuk a $Q(x, y)$ pont koordinátáit, $Q'(x, y)$ pontot el kell forgatni – az óra mutató járásával megegyező irányban – $\alpha/2$ szöggel. Ezt a következő képletek alkalmazásával tehetjük meg:

$$Q.x = (Q'.x - O.x) \cdot \cos\left(\frac{\alpha}{2}\right) + (Q'.y - O.y) \cdot \sin\left(\frac{\alpha}{2}\right) + O.x$$

$$Q.y = -(Q'.x - O.x) \cdot \sin\left(\frac{\alpha}{2}\right) + (Q'.y - O.y) \cdot \cos\left(\frac{\alpha}{2}\right) + O.y ,$$

ahol $Q.x$ és $Q.y$ jelöli a $Q(x, y)$ pont x és y koordinátáit. Ezzel analóg $Q'.x$, $Q'.y$, $O.x$ és $O.y$ koordináták jelölése is.

Most általánosítsuk a kapott képleteket úgy, hogy az adott síkidom minden csúcsának koordinátáit meg tudjuk határozni velük! Ehhez mindössze a szögfüggvényekben szereplő elforgatási szögeket kell kiegészíteni a következő értékkel: $k \cdot \alpha$, ahol $k = 0, 1, \dots, n-1$ (a példában $n=5$ -re: $k = 0, 1, \dots, 4$). Ekkor a következőképpen változnak meg a képletek:

$$Q_{k+1}.x = (Q'.x - O.x) \cdot \cos\left(\frac{\alpha}{2} + k \cdot \alpha\right) + (Q'.y - O.y) \cdot \sin\left(\frac{\alpha}{2} + k \cdot \alpha\right) + O.x$$

$$Q_{k+1}.y = -(Q'.x - O.x) \cdot \sin\left(\frac{\alpha}{2} + k \cdot \alpha\right) + (Q'.y - O.y) \cdot \cos\left(\frac{\alpha}{2} + k \cdot \alpha\right) + O.y ,$$

ahol $k = 0, 1, \dots, n-1$ továbbra is, $Q_{k+1}.x$ és $Q_{k+1}.y$ jelöli a síkidom megfelelő csúcsainak x és y koordinátáit.

5. Most, hogy már tudunk hivatkozni az adott síkidom bármely csúcsára, minimumkereséssel megkeressük a csúcsai közül a $P(x, y)$ ponthoz legközelebb lévő. Ezt a maximumkeresés programozási tétel átírásával tehetjük meg. Ehhez azonban a ciklus minden lépésében meg kell határozni az aktuális távolságot a $P(x, y)$ pont és az aktuális csúcs között. Mivel minden adat ismert, ezért ezt egy lépésben megtehetjük:

$$d_{akt} = \sqrt{(Q_{akt}.x - P.x)^2 + (Q_{akt}.y - P.y)^2} ,$$

ahol d_{akt} jelöli az aktuális távolságot, $Q_{akt}.x$ és $Q_{akt}.y$ pedig az aktuális csúcs x és y koordinátáit.

A minimumkeresés eredményeként megkapjuk a keresett „d” távolságot, mely – a feladat távolság-definíciója szerint a $P(x, y)$ pont távolságát határozza meg az adott síkidomtól számítva.

Függvények:

Azért, hogy az eddig kiszámított képleteket ne kelljen a továbbiakban újra és újra leírni, most bevezetjük a következő függvényeket:

$Csucs(k+1) = Q_{k+1}(x, y)$, ahol:

$$Q_{k+1}.x = (Q'.x - O.x) \cdot \cos\left(\frac{\alpha}{2} + k \cdot \alpha\right) + (Q'.y - O.y) \cdot \sin\left(\frac{\alpha}{2} + k \cdot \alpha\right) + O.x$$

$$Q_{k+1}.y = -(Q'.x - O.x) \cdot \sin\left(\frac{\alpha}{2} + k \cdot \alpha\right) + (Q'.y - O.y) \cdot \cos\left(\frac{\alpha}{2} + k \cdot \alpha\right) + O.y$$

$$k = 0, 1, \dots, n-1$$

A szabályos síkidomokkal kapcsolatos számítások 4. pontjának megfelelően, n pedig változatlanul a síkidom oldalszámát jelöli.

$$\text{Ponttavolsag}(P(x, y), Q(x, y)) = \sqrt{(Q.x - P.x)^2 + (Q.y - P.y)^2}$$

$\text{Tavolsag}(P(x, y), S) = d$, ahol:

$$d = \sqrt{(O.x - P.x)^2 + (O.y - P.y)^2} - r \quad (r \text{ a kör sugara, } O(x, y) \text{ pedig középpontja), ha az „S” síkidom kör;}$$

$$d = \text{Ponttavolsag}(P(x, y), \text{Csucs}(i)) \wedge \forall j \in [1..n]:$$

$\text{Ponttavolsag}(P(x, y), \text{Csucs}(j)) \geq \text{Ponttavolsag}(P(x, y), \text{Csucs}(i))$ (vagyis $\text{Csucs}(i)$ az $P(x, y)$ ponthoz legközelebbi csúc), ha az „S” síkidom tetszőleges, n oldalú, szabályos sokszög.

$\text{Sokszog}(S) = l$, ahol:

$$l = \text{igaz} \mid n \geq 3$$

$$l = \text{hamis} \mid n < 3$$

Az „ n ” változó továbbra is az adott sokszög oldalszámát jelöli. Amennyiben $n < 3$, és adott a síkidom köré írt kör sugara, akkor – definíció szerint – a vizsgált síkidomot körnek tekintjük. Kör esetén természetesen a síkidom köré írt kör maga a síkidom.

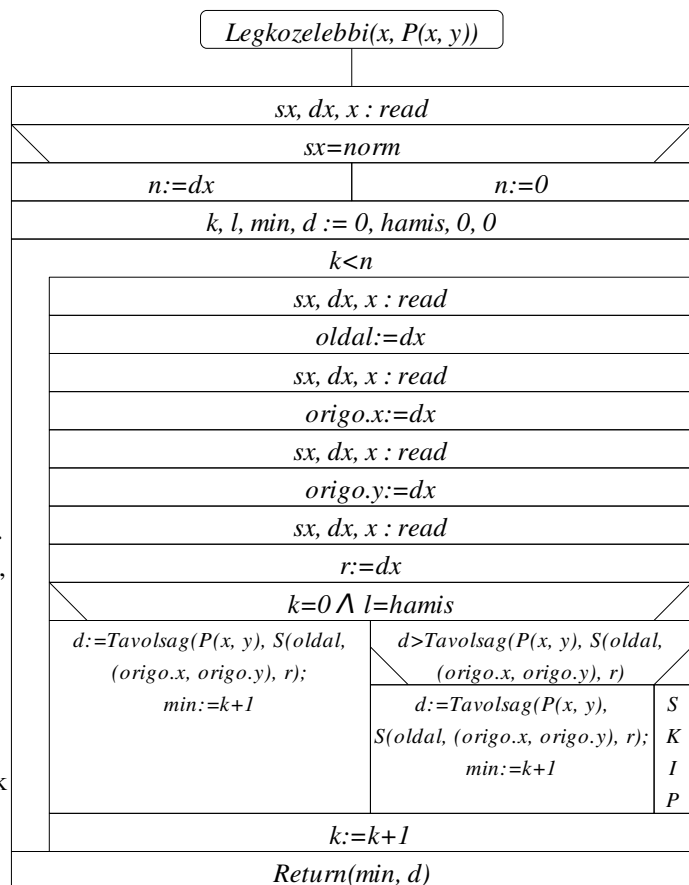
Megoldás:

A feladat megoldását adó absztrakt programot a 3. ábrán láthatjuk. A program függvényként kerül meghívásra, a függvény eredményeként a $P(x, y)$ ponthoz legközelebbi objektum sorszámát, illetve a $P(x, y)$ ponttól való távolságot kapjuk.

A függvény bemenő paraméterei az „ x ” szekvenciális fájl, illetve a $P(x, y)$ pont.

A távolság kiszámításához minden esetben két alapvető adat kell: a $P(x, y)$ pont, illetve az aktuális síkidom adatai. A síkidom minden esetben három alapvető adattal kerül vizsgálatra, függetlenül attól, hogy éppen kört vagy szabályos sokszöget vizsgálunk. Előbbi esetben a síkidom harmadik adata a kör sugara, utóbbi esetben pedig a síkidom oldalának hossza. Az absztrakt programban használt függvények úgy lettek definiálva, hogy mindkét esetre jól működjenek.

Amennyiben üres fájlból próbálunk meg adatokat gyűjteni, akkor a program eredménye abnormalis érték lesz: a legközelebbi objektum sorszáma 0 lesz, ugyanígy a $P(x, y)$ pont síkidomtól mért távolsága is. Ez normális esetben (vagyis legalább 1 adatot tartalmazó fájl esetén) nem fordul elő. Ennek a



ábra 3: Legközelebbi síkidom megkeresése

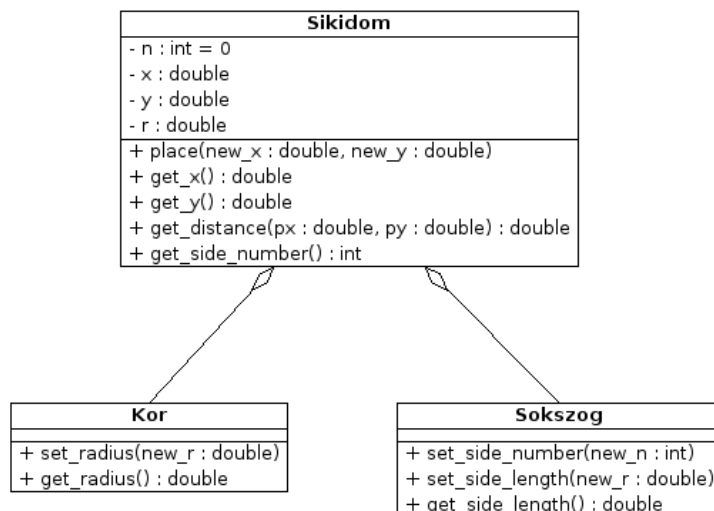
hibának a kezelésére a tényleges program megírásakor kerül sor (az absztrakt program áttekinthetőségének érdekében).

Osztálydiagramok:

A síkidomokat egy közös, „Sikidom” nevű osztállyal reprezentáljuk, melyben letárolásra kerülnek az egyes síkidomok oldalszámai (n), koordinátái (x, y), illetve a sugár (amennyiben a vizsgált síkidom kör), vagy az oldalhossz (sokszögek esetén), melynek értéke az „r” változóban kerül letárolásra.

Az egyes síkidomokat (kör, sokszög) a „Sikidom” nevű osztályból származtatjuk. Az egyes síkidomokhoz a fent említett osztály, illetve az egyes síkidom-osztályok saját függvényein és eljárásain keresztül férhetünk hozzá.

A „Kor” és a „Sokszog” osztályok származtatása a „Sikidom” osztályból



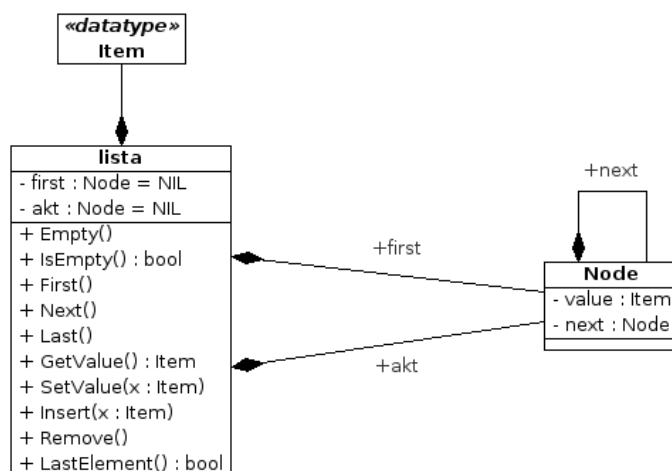
ábra 4: A síkidomok osztálydiagramja

A síkidomok osztályai között fennálló kapcsolatokat a 4. ábrán láthatjuk egy szabványos, UML diagram formájában.

A konkrét megoldóprogramban először feltöltünk egy láncolt listát (a feladatkiírásnak megfelelően), majd ebből a listából visszaolvasva számítjuk ki az egyes síkidomok távolságait egy adott P(x, y) ponttól. A láncolt lista adatszerkezetét az 5. ábrán látható UML diagram szemlélteti.

Megjegyezzük, hogy az ábrán látható „Item” adattípus a szóban forgó adatszerkezet általánosítása (sablonja, vagy template), hogy az adatszerkezet – gyakorlatilag – az összes ismert adatszerkezetre működjön. Jelen esetben az „Item” adattípus értéke a „Sikidom” osztály.

A „lista” adatszerkezet felépítése a „Node” struktúra segítségével és az „Item” adattípus felhasználásával



ábra 5: A láncolt lista adatszerkezet osztálydiagramja

Reprezentáció:

Ahogy az az ábrákon is jól látszik, a reprezentáció szintjén nem teszünk különbséget az egyes síkidomok fajtái között. Függetlenül attól, hogy az éppen vizsgált síkidom a körök (Kor) vagy a sokszögek (Sokszog)

osztályába tartozik, minden esetben 4 alapvető adattal reprezentáljuk őket:

1. A síkidom oldalainak száma: alapvetően ez dönti el, hogy a vizsgált síkidom a körök vagy a sokszögek osztályába kerül besorolásra. Ezt az értéket az „n” változóban tároljuk. A választott kódolási konvenció alapján, ha $n < 3$, akkor a síkidom a körök osztályába lesz besorolva, ellenkező esetben pedig a sokszögekébe. A megfelelő számítási műveletek is ennek a besorolásnak alapján lesznek elvégezve;
2. A középpont „x” koordinátája: a koordináta-értékét a „double” típusú „x” változóban tároljuk;
3. A középpont „y” koordinátája: az „x” koordinátával analóg módon tároljuk;
4. A sugár – vagy oldal – hossza: attól függően, hogy a síkidom a körök vagy a sokszögek osztályába tartozik, beszélhetünk a síkidom sugarának vagy oldalának hosszáról. Reprezentációs szinten nincs különbség a kétféle adat között, mindkettőt a „double” típusú „r” változóban tároljuk.

Azt, hogy egy konkrét objektum esetén az „r” változóban tárolt értéket sugárként vagy oldalhosszként kezeljük, az objektum osztály-besorolása dönti el. Ennek akkor van jelentősége, amikor a síkidom az adott $P(x, y)$ ponttól való távolságát számítjuk ki.

A távolság meghatározása a feladat szövegében definiált módon történik, akár sugárként, akár oldalhosszként tekintünk az „r” változóra, mindkét esetben a neki megfelelő módon történik a távolság kiszámítása. Ezt az „n” változó értékének ismerete garantálja.

Forráskód – fejáallomány (Sikidom.h):

```
#ifndef SIKIDOM_H
#define SIKIDOM_H
#include <iostream>
#include <math.h>

//SIKIDOMOK OSZTALYA
class Sikidom{
protected:
    double x, y, r;
    int n;
public:
    Sikidom(){ n=0; }
    void place(const double new_x, const double new_y);
    double get_x();
    double get_y();
    double get_distance(const double px, const double py);
    int get_side_number();
};

//KOROK OSZTALYA
class Kor : public Sikidom{
public:
    void set_radius(const double new_r);
    double get_radius();
};

//SOKSZOGEK OSZTALYA
class Sokszog : public Sikidom{
public:
    void set_side_number(const int new_n);
    void set_side_length(const double new_r);
    double get_side_length();
};

#endif
```

Forráskód – megvalósítás (Sikidom.cpp):

```

#include <math.h>
#include "Sikidom.h"

//SIKIDOM ELHELYEZÉSE
void Sikidom::place(const double new_x, const double new_y){
    x = new_x;
    y = new_y;
}

//ORIGO KEKERDEZÉSE
double Sikidom::get_x(){ return x; }
double Sikidom::get_y(){ return y; }

/*
 * TAVOLSAG MEGHATÁROZÁSA
 *
 * Hasznalat:
 * a.get_distance(px, py);
 *
 * Információk:
 * a - Sikidom;
 * px, py - integer;
 * A függvény meghatározza a kor távolságát egy adott
 * p ponttól, melyet a px es py paraméterekben megadott
 * koordinátákkal határozunk meg.
 */
double Sikidom::get_distance(const double px, const double py){
    double dist=0;
    if(n==0){
        double dx=x-px;
        double dy=y-py;
        dist=sqrt((dx*dx)+(dy*dy))-r;
    }else{
        double alpha=(asin(1)*4)/n;
        double rad=r/(sin(alpha/2)*2);
        double qx0=x;
        double qy0=y-rad;
        double qx1, qy1;
        double qx, qy;
        double dist0;
        for(int i=0; i<n; ++i){
            qx1=((qx0-x)*cos((alpha/2)+(i*alpha)))+(qy0-y)*sin((alpha/2)+(i*alpha))+x;
            qy1=(-1*((qx0-x)*sin((alpha/2)+(i*alpha)))+(qy0-
y)*cos((alpha/2)+(i*alpha)))+y;
            qx=qx1-px;
            qy=qy1-py;
            dist0=sqrt((qx*qx)+(qy*qy));
            if(i==0){
                dist=dist0;
            }else{
                if(dist0<dist){
                    dist=dist0;
                }
            }
        }
        return dist;
    }
}

//OLDALSZAM LEKERDEZÉSE
int Sikidom::get_side_number(){ return n; }

//SUGAR BEALLÍTÁSA
void Kor::set_radius(const double new_r){
    r = new_r;
}

//SUGAR LEKERDEZÉSE
double Kor::get_radius(){
    return r;
}

//OLDALSZAM BEALLÍTÁSA

```

```

void Sokszog::set_side_number(const int new_n){
    n = new_n;
}
//OLDALHOSSZ BEALLITASA
void Sokszog::set_side_length(const double new_r){
    r = new_r;
}
//OLDALHOSSZ LEKERDEZESE
double Sokszog::get_side_length(){ return r; }

```

Forráskód – fejláblomány (lista.h):

```

#ifndef LISTA_H
#define LISTA_H
#include <iostream>

/*
 * LANCOLT LISTA
 *
 * Az osztaly az "Algoritmusok es Adatszerkezetek" targyban
 * tanultak alapjan lett elkeszítve, ez az osztaly tagfuggvenyeire
 * is vonatkozik.
 */
template <class Item>
class lista{
private:
    struct Node{
        Item value;
        Node *next;
    };
    Node* first;
    Node* akt;
public:
    //KONSTRUKTOR
    lista(){
        first=NULL;
        akt=NULL;
    }
    //DESTRUKTOR
    ~lista(){
        Node* p;
        while(first!=NULL){
            p=first;
            first=first->next;
            delete p;
        }
        first=NULL;
        akt=NULL;
    }
    //COPY-KONSTRUKTOR
    lista(const lista<Item>& s);
    //ERTEKADAS-OPERATOR
    lista<Item>& operator=(const lista<Item>& s);
    //KIVETELKEZELES
    enum Exceptions{EMPTYLIST, NONEXT, AEMPTY, WRELEM};
    //ELJARASOK ES FUGGVENYEK
    void Empty();
    bool IsEmpty();
    void First();
    void Next();
    void Last();
    Item GetValue();
    void SetValue(const Item x);
    void Insert(const Item x);
    void Remove();
    bool LastElement();
};

#endif

```


Forráskód – megvalósítás (lista.cpp):

```
#include <iostream>

//LISTA KIURITESE
template <class Item>
void lista<Item>::Empty(){
    if(first!=NULL){
        akt=first;
        Node* p;
        while(akt!=NULL){
            p=akt;
            akt=akt->next;
            delete p;
        }
        first=NULL;
        akt=NULL;
    }else{
        throw AEMPTY;
    }
}

//URESSEG LEKERDEZESE
template <class Item>
bool lista<Item>::IsEmpty(){
    return first==NULL;
};

//UGRASS A LISTA ELEJERE
template <class Item>
void lista<Item>::First(){
    if(first!=NULL){
        akt=first;
    }else{
        throw EMPTYLIST;
    }
}

//UGRASS A KOVETKEZO ELEMRE
template <class Item>
void lista<Item>::Next(){
    if(akt->next!=NULL){
        akt=akt->next;
    }else{
        throw NONEXT;
    }
}

//UGRASS AZ UTOLSO ELEMRE
template <class Item>
void lista<Item>::Last(){
    if(first==NULL){
        throw EMPTYLIST;
    }else{
        if(akt==NULL){
            akt=first;
        }
        while(akt->next!=NULL){
            akt=akt->next;
        }
    }
}

//ELEM LEKERDEZESE
template <class Item>
Item lista<Item>::GetValue(){
    if(akt!=NULL){
        return akt->value;
    }else{
        throw WRELEM;
    }
}

//ELEM BEALLITASA
template <class Item>
void lista<Item>::SetValue(const Item x){
    if(akt!=NULL){
```

```

        akt->value=x;
    }else{
        throw WRELEM;
    }
}
//ELEM BESZURASA
template <class Item>
void lista<Item>::Insert(const Item x){
    if((akt==NULL)&&(first!=NULL)){
        throw WRELEM;
    }else{
        Node* p;
        p = new Node();
        p->value=x;
        p->next=NULL;
        if(first!=NULL){
            p->next=akt->next;
            akt->next=p;
        }else{
            first=p;
        }
        akt=p;
    }
}
//ELEM ELTAVOLITASA
template <class Item>
void lista<Item>::Remove(){
    if(akt!=NULL){
        if(akt!=first){
            Node* p=first;
            while(p->next!=akt){
                p=p->next;
            }
            p->next=akt->next;
            delete(akt);
            akt=p->next;
        }else{
            first=first->next;
            delete(akt);
            akt=first;
        }
    }else{
        throw WRELEM;
    }
}
//POZICIO LEKERDEZESE
template <class Item>
bool lista<Item>::LastElement(){
    if(akt!=NULL){
        if(akt->next==NULL){
            return true;
        }else{
            return false;
        }
    }else{
        throw WRELEM;
    }
}
}

```

Forráskód – főprogram (kgeom.cpp):

```

/*
 * HARMADIK SZAMU BEADANDO
 * Keszitette: Abraham Robert
 * EHA-kod: ABROAAI.ELTE
 * Datum: 2008/11/27.
 */

#include <iostream>
#include <cstdlib>

```

```

#include <string>
#include <fstream>
#include "Sikidom.h"
#include "Sikidom.cpp"
#include "lista.h"
#include "lista.cpp"

using namespace std;

//Legyen egyszerubb a beolvasas (Standard Input)!
void ChooseBetween(    const string Kerdes,
                      const string OPT1, const char OPT1_Key,
                      const string OPT2, const char OPT2_Key,
                      char& Selected);

template <class Item>
void SetData(const string Kerdes, const Item Minimum, Item& Adat);

int main()
{
    //ALAPVALTOZOK
    char datain;
    int n;
    double px, py;
    string fname;
    //MENU
    cout << "KOORDINATAGEOMETRIA" << endl;
    cout << endl;
    cout << "A program megkeresi egy adott P ponthoz legkozelebb levo sikidomot" << endl;
    cout << "A sikidomok beolvasasa tortenhet fajlbol es billentyuzetrol is." << endl;
    cout << endl;
    cout << "A P pont koordinatai:" << endl;
    cout << "x: "; cin >> px;
    cout << "y: "; cin >> py;
    ChooseBetween("Sikidomok beolvasasa", "Fajlbol", 'F', "Billentyuzetrol", 'B', datain);
    //BEOLVASAS-SZAMITAS
    const int min=1;
    double x, y, r;
    int sn;
    const int osmin=3;
    bool Circle;
    bool FileMode;
    //Billentyurol
    if(datain=='b'){
        SetData<int>("Sikidomok szama", min, n);
        FileMode=false;
    }else{
        //Fajlbol
        cout << endl;
        cout << "Fajlnev: "; cin >> fname;
        FileMode=true;
    }
    //Beolvasas
    ifstream inp(fname.c_str());
    if(FileMode==true){
        if(inp.fail()){
            cout << endl;
            cout << "Hiba a fajl megnyitasakor!" << endl;
            exit(1);
        }
        inp >> n;
        cout << endl;
        cout << "Sikidomok szama: " << n << endl;
    }
    //LISTAKESZITES
    lista<Sikidom> l;
    for(int i=0; i<n; ++i){
        cout << endl;
        cout << i+1 << ". SIKIDOM ADATAI" << endl;
        if(FileMode==false){
            //Tipus megadasa
            char gmttype;
            ChooseBetween("Tipus", "Kor", 'K', "Sokszog", 'S', gmttype);

```

```

        if(gmtype=='k'){
            //Sugar megadása
            SetData<double>("Sugar hossza", 0, r);
            Circle=true;
        }else{
            //Oldalszam megadása
            SetData<int>("Oldalszam", osmin, sn);
            //Oldalhossz megadása
            SetData<double>("Oldal hossza", 0, r);
            Circle=false;
        }
        cout << endl;
        cout << "Koordinatak:" << endl;
        cout << "x: "; cin >> x;
        cout << "y: "; cin >> y;
    }else{
        //Beolvasas fajlbol
        inp >> sn;
        inp >> x;
        inp >> y;
        inp >> r;
        cout << endl;
        if(sn>=3){
            Circle=false;
            cout << "Oldalszam: " << sn << endl;
            cout << "Oldalak hossza: " << r << endl;
        }else{
            Circle=true;
            cout << "Sugar hossza: " << r << endl;
        }
        cout << "Koordinatak: (" << x << ", " << y << ")" << endl;
    }
    //Beszuras
    if(Circle=true){
        Kor K;
        K.place(x, y);
        K.set_radius(r);
        l.Insert(K);
    }else{
        Sokszog S;
        S.place(x, y);
        S.set_side_number(sn);
        S.set_side_length(r);
        l.Insert(S);
    }
}
//BEOLVASAS LISTABOL
l.First();
Sikidom Item=l.GetValue();
int closest=0;
double distance=Item.get_distance(px, py);
double dist0=0;
int i=0;
//Minimumkereses
while(l.LastElement()==false){
    ++i;
    l.Next();
    Item=l.GetValue();
    dist0=Item.get_distance(px, py);
    if(dist0<distance){
        distance=dist0;
        closest=i;
    }
}
//EREDMENYEK KIIRASA
cout << endl;
cout << "A P(" << px << ", " << py << ") ponthoz legkozelebbi objektum:" << endl;
cout << "A lista " << closest+1 << ". objektuma." << endl;
cout << "Tavolsag: " << distance << " egyseg." << endl;
return 0;
}

```

```
void ChooseBetween(    const string Kerdes,
                      const string OPT1, const char OPT1_Key,
                      const string OPT2, const char OPT2_Key,
                      char& Selected){
    do{
        cout << endl;
        cout << Kerdes << ":" << endl;
        cout << OPT1 << " [" << OPT1_Key << "]" << endl;
        cout << OPT2 << " [" << OPT2_Key << "]" << endl;
        cout << endl;
        cout << "Valasztas: "; cin >> Selected;
        Selected=tolower(Selected);
    }while((Selected!=tolower(OPT1_Key))&&(Selected!=tolower(OPT2_Key)));
}

template <class Item>
void SetData(const string Kerdes, const Item Minimum, Item& Adat){
    do{
        cout << endl;
        cout << Kerdes << " (min. " << Minimum << "): "; cin >> Adat;
        if(Adat<Minimum){
            cout << endl;
            cout << Minimum << " a megadható legkisebb érték!" << endl;
        }
    }while(Adat<Minimum);
}
```

Tesztelés:

- Menü kipróbálása (beolvasási módok tesztelése);
- Billentyűről olvasás tesztelése (objektumok létrehozási kísérlete normális és szélsőséges adatokkal is);
- Fájlból olvasás tesztelése (nem létező fájlal);
- Fájlból olvasás tesztelésének folytatása (létező fájlból adatok betöltése);
- Eredmények áttekintése, leellenőrzése egyszerű, konkrét példákon keresztül;

Fejlesztési lehetőségek:

- A ChooseBetween(...) eljárás kiterjesztése tetszőlegesen hosszú listára;
- A SetData< ... > eljárás kiterjesztése karakterláncokra, illetve stringekre;
- A „lista.h” modulban definiált, és „lista.cpp” modulban megvalósított láncolt lista indexelhetőségének megoldása, ennek megfelelően az értékadó operátor és copy-konstruktor megfelelő felüldefiniálása;