

Objektumok példányosítása

Objektum kettős jelentése

Modellezési szempontból

- ❑ Az objektum a megoldandó problémának egy **egyediként azonosítható** része.
- ❑ Az objektum elrejt a felelősségi köréhez tartozó adatokat: azokat kizárólag a metódusai révén kezeljük (olvassuk, módosítjuk).
- ❑ Egy objektum életciklusa annak létrejöttével kezdődik el, és a megszűnésével áll le.

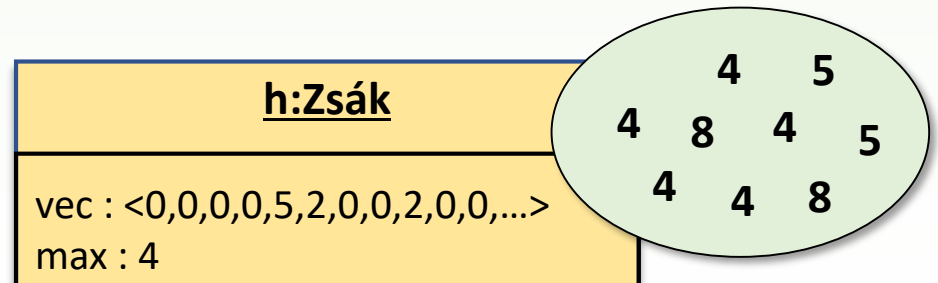
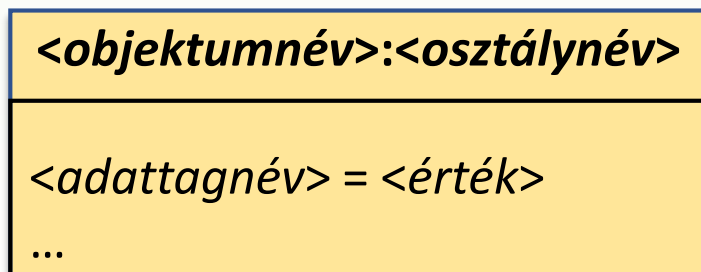
Programnyelvi megközelítés

- ❑ Az objektum az a **memória foglалás**, ahol az objektum az adatait tárolja.
- ❑ Az objektum memóriaterületén tárolt értékek láthatósági köre szabályozható, de az objektum metódusai mindig elérik azokat.
- ❑ Egy objektum memória-foglalását (példányosítását) konstruktora, törlését a destruktora végzi.

Objektum UML jelölése

□ Egy objektumot meghatároz

- az **osztálya**, amely azon objektumok halmaza, amelyek ugyanolyan adattagokkal és metódusokkal rendelkeznek. Az osztály leírja
 - név-típus párok formájában az objektum **adatait** (tulajdonságait, attribútumait, mezőit) és
 - az objektumra meghívható (az objektum adattagjainak értékeit manipuláló) **metódusokat** (tagfüggvényeket, műveleteket).
- a **neve** (amit nem kötelező megadni),
- az **állapota** (amit az adattagjainak értéke határoz meg).



Feladat

Készítsünk programot, amely feltölt egy tömböt különféle sokszögekkel, majd mindegyiket eltolja ugyanazon mértékkel, és kiszámolja az így nyert sokszögek súlypontjait. A sokszögek csúcspontjainak és súlypontjának koordinátái, sőt az eltolás koordinátái is legyenek egész számok.

Objektumok:

- sokszögek
- síkbeli pontok, amelyek a sokszögek csúcsait és súlypontjukat adják
- egy tömbben tároljuk a sokszögeket

Single responsibility

O
L
I
D

Objektumok felelősségi köre:

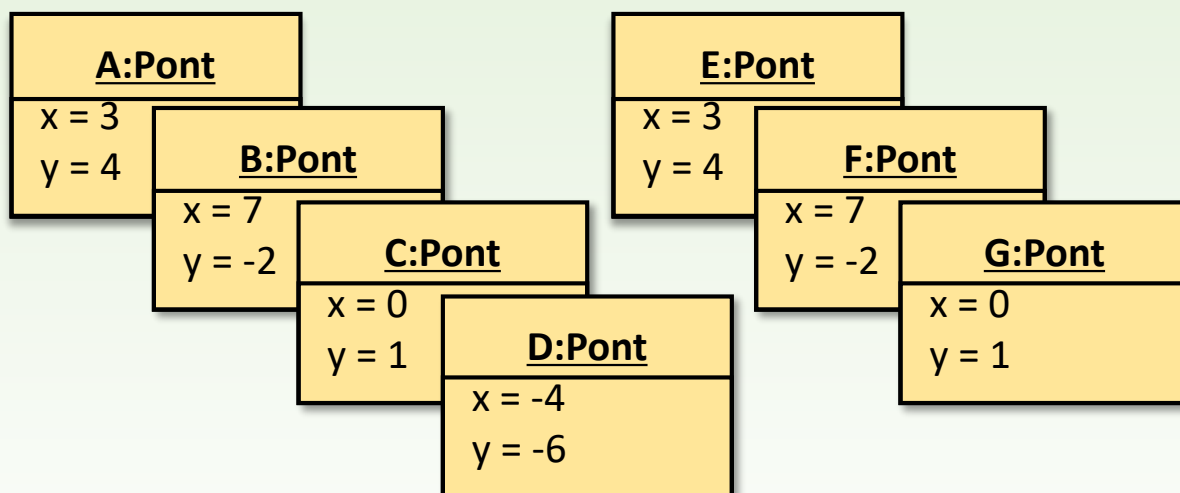
- sokszögek: eltolása, súlypontjának kiszámítása, csúcsai számának, adott csúcspontjának lekérdezése és módosítása
- síkbeli pontok: eltolása, koordinátáinak lekérdezése és módosítása
- tömb: adott indexű elem elérése, hosszának lekérdezése

A feladat objektumai

Pontok osztálya:

Pont
x : int
y : int
eltol()

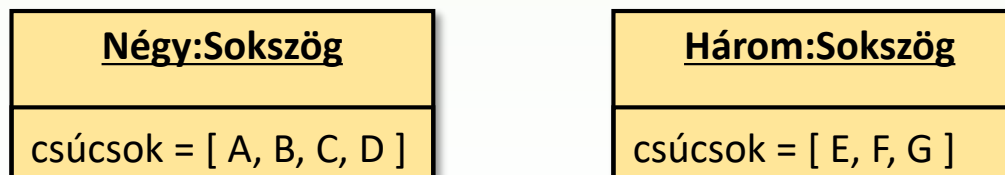
Pont objektumok:



Sokszögek osztálya:

Sokszög
csúcsok : Pont[]
eltol()
súlypont()

Sokszög objektumok:



Osztályleírás részletezettsége

- Az osztály-leírás **a modellezés során fokozatosan alakul ki**, ezért a modellezés adott szintjén bizonyos részletei még hiányozhatnak.
- Hiányozhatnak belőle vagy egyáltalán nem szerepelnek még benne attribútumok és/vagy a metódusok.
 - Hiányozhat az adattagok típusa, metódusok paraméterezése és a visszatérési típusa.
 - Nincsenek még feltüntetve a láthatósági jelölések.

<osztálynév>

<osztálynév>

<metódusnév>()

<osztálynév>

- <adattagnév> : <típusnév>

+ <metódusnév>(<paraméterek>) : <típusnév>

Osztályleírás kiegészítései

- ❑ **Megszorításokat** írhatunk elő mind az adattagok értékére, mind a metódusok működésére az azok mellé írt {...} jelzésben. (Pl. az adattagokat nem módosító műveleteket a {query} jelzi.)
- ❑ **Kezdőértéket** adhatunk meg az adattagokhoz (a konstruktor állítja be).
- ❑ Később kitöltendő **paramétereket** adhatunk az osztályhoz.
- ❑ Viselkedési tulajdonságokat jelölhetünk ki a <<...>> jelzéssel magára az osztályra (pl. <<interface>> , <<enumeration>>), vagy metódusainak illetve adattagjainak egy-egy csoportjára (pl. <<getter>> , <<setter>>).
 - A rejtett adattagok értékének ellenőrzött **módosítását** (setter), illetve **lekérdezését** (getter) publikus műveletekkel szokás biztosítani.

<osztálynév>	
- <adattagnév> : <típus> = <kezdőérték> { <feltétel> }	<paraméter>
+ <metódusnév>(<paraméterlista>) : <típus> {query}	

<<interface>> <osztálynév>	
<<csoport>> - <adattagnév> : <típus>	
<<getter>> + <metódusnév>() : <típus>	

A feladat modellezésének szintjei

Elemzés szintje

Pont
x : int y : int
eltol()

Tervezési döntések:

- legyenek az adattagok **publikusak**
- az eltolás paraméterként kapja meg az eltolás mértékét megadó pontot, és ne változtassa meg azt a pontot, amire meghívják, hanem újat hozzon létre:

c = a.eltol(b)

// c az a-hoz képest b-vel eltolt pont

Tervezés szintje

Pont
+ x : int + y : int
+ eltol(mp:Pont) : Pont {query}

Pont c;
c.x = x + mp.x;
c.y = y + mp.y;
return c;

return Pont(x + mp.x, y + mp.y)

return Pont(x / f, y / f)

Megvalósítás szintje

Pont
+ x : int + y : int
+Pont(int,int)
+Pont()
+ setPont(x:int, y:int)
+eltol(mp:Pont) : Pont {query}
+operator+(mp:Pont) : Pont {query}
+operator/(f:int) : Pont {query}

legyen kétféle konstruktor

setter

c = a.eltol(b)
helyett:
c = a + b

új művelet a súlypont kiszámításához

Pont osztály C++ kódja

```
class Point
```

```
{
```

```
    public:
```

```
        Point(int x = 0, int y = 0): _x(x), _y(y) { }
```

```
        void setPoint(int x, int y) { _x = x; _y = y; }
```

```
        Point move(const Point &mp) const
        { return Point(_x + mp._x, _y + mp._y); }
```

```
        Point operator+(const Point &mp) const
        { return Point(_x + mp._x, _y + mp._y); }
```

```
        Point operator/(int f) const
        { return Point(_x / f, _y / f); }
```

```
    public:
```

```
        int _x, _y;
```

```
};
```

paraméterváltozók default értéke:

```
Point a;
Point b(3);
Point c(-4, 8);
```

adattagok inicializálása

query

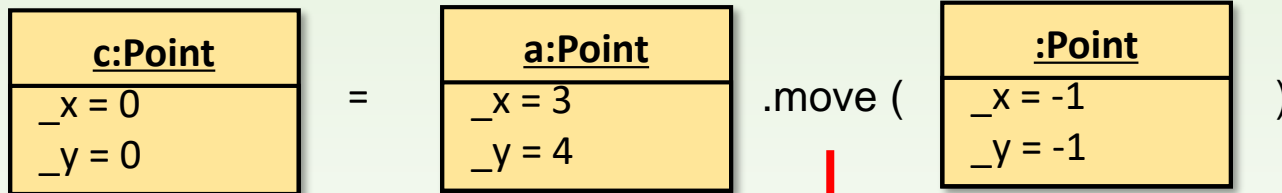
operátor felüldefiniálás

„inline” definíciók

Pont objektumok példányosítása

```
Point a(3,4);  
Point c;  
c = a.move(Point(-1,-1));
```

move() hívása előtt:



Erre a pontra hivatkozik a move() metódus mp paraméterváltozója.

move() hívása közben:

move() hívása után:



Ez a move()-ban ideiglenesen létrehozott pont, amelynek koordinátáit az a és az mp alapján számoljuk.

Minden objektum rendelkezik egy értékadás operátorral.

```
Point move(const Point &mp) const {  
    return Point(_x + mp._x, _y + mp._y );  
}
```

Objektum default pointere

```
class Point{  
public:  
    Point(int x=0, int y=0):_x(x),_y(y) {}  
  
    void setPoint(int x, int y) { _x = x; _y = y; }  
    void setPoint(int x, int y) { this->_x = x; this->_y = y; }  
  
    ...  
  
public:  
    int _x, _y;  
};
```

A **this** egy alapértelmezett módon létező pointerváltozó, amely azon objektumnak a memória címét tartalmazza (arra mutat), amely objektumra a metódust meghívják: ez p.setPoint(3, -2) esetén a p.

3

Az objektum orientáltság további ismérve a **nyílt rekurzió**: az objektum mindig látja saját magát, eléri műveleteit és adatait.

Sokszög osztály tervezése

Elemzés szintje

Sokszög
csúcsok : Pont[]
eltol() súlypont()

Tervezési döntések:

- adattag legyen privát
- az eltolás magát a sokszög objektumot, azaz annak csúcsait tolja el

Tervezés szintje

Sokszög
- csúcsok : Pont[]
+ eltol(mp:Pont) : void + súlypont() : Pont {query}

```
for i=1 .. oldalszám() loop
    csúcsok[i] = csúcsok[i] + mp
endloop
```

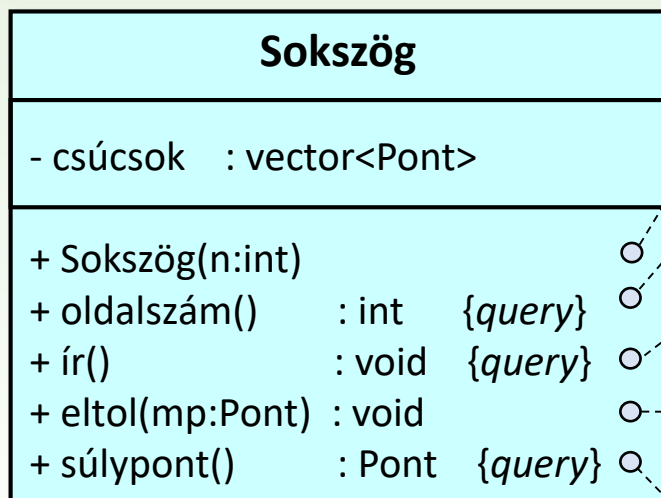
```
Pont sp;
for i=1 .. csúcsok.size() loop
    sp := sp + csúcsok[i]
endloop
return sp / oldalszám();
```

Megvalósítás szintje

Sokszög
- csúcsok : vector<Pont>
+ Sokszög(n:int) a csúcsok száma + oldalszám() : int {query} getter + ír() : void {query} + eltol(mp:Pont) : void kírás + súlypont() : Pont {query}

Sokszög osztály

Megvalósítás szintje



Olyan ciklus (foreach), amelyik bejárja egy gyűjtemény elemeit (de nem változtathatja meg).

hibaellenőrzés

```
if n < 3 then hiba endif  
csúcsok.resize(n)
```

```
return csúcsok.size()
```

```
forall csúcs in csúcsok loop  
  ír(csúcs.x); ír(csúcs.y)  
endloop
```

```
for i=0 .. csúcsok.size()-1 loop  
  csúcsok[i] = csúcsok[i] + mp  
endloop
```

```
Pont sp;  
forall csúcs in csúcsok loop  
  sp := sp + csúcs  
endloop  
return sp / oldalszám()
```

Sokszög osztály kódja

```
class Polygon
{
public:
    enum Errors{FEW_VERTICES};

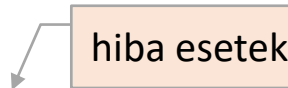
    Polygon(int n) : _vertices(n) {
        if (n < 3) throw FEW_VERTICES;
    }

    unsigned int sides() const { return _vertices.size(); }

    void write() const;

    void move(const Point &mp);

    Point center() const;
private:
    std::vector<Point> _vertices;
};
```



hiba esetek

Sokszög többi metódusának kódja

```
void Polygon::move(const Point &mp)
```

```
{
```

```
    for(unsigned int i=0; i<_vertices.size(); ++i) {
```

```
        _vertices[i] = _vertices[i] + mp;
```

```
    }
```

```
}
```

```
Point Polygon::center() const
```

```
{
```

```
    Point center;
```

```
    for(Point vertex : _vertices) {
```

```
        center = center + vertex;
```

```
    }
```

```
    return center / sides();
```

```
}
```

```
void Polygon::write() const
```

```
{
```

```
    cout << "<";
```

```
    for( Point vertex : _vertices ){
```

```
        cout << "(" << vertex._x  
            << ", " << vertex._y << ")";
```

```
    }
```

```
    cout << ">\n";
```

```
}
```

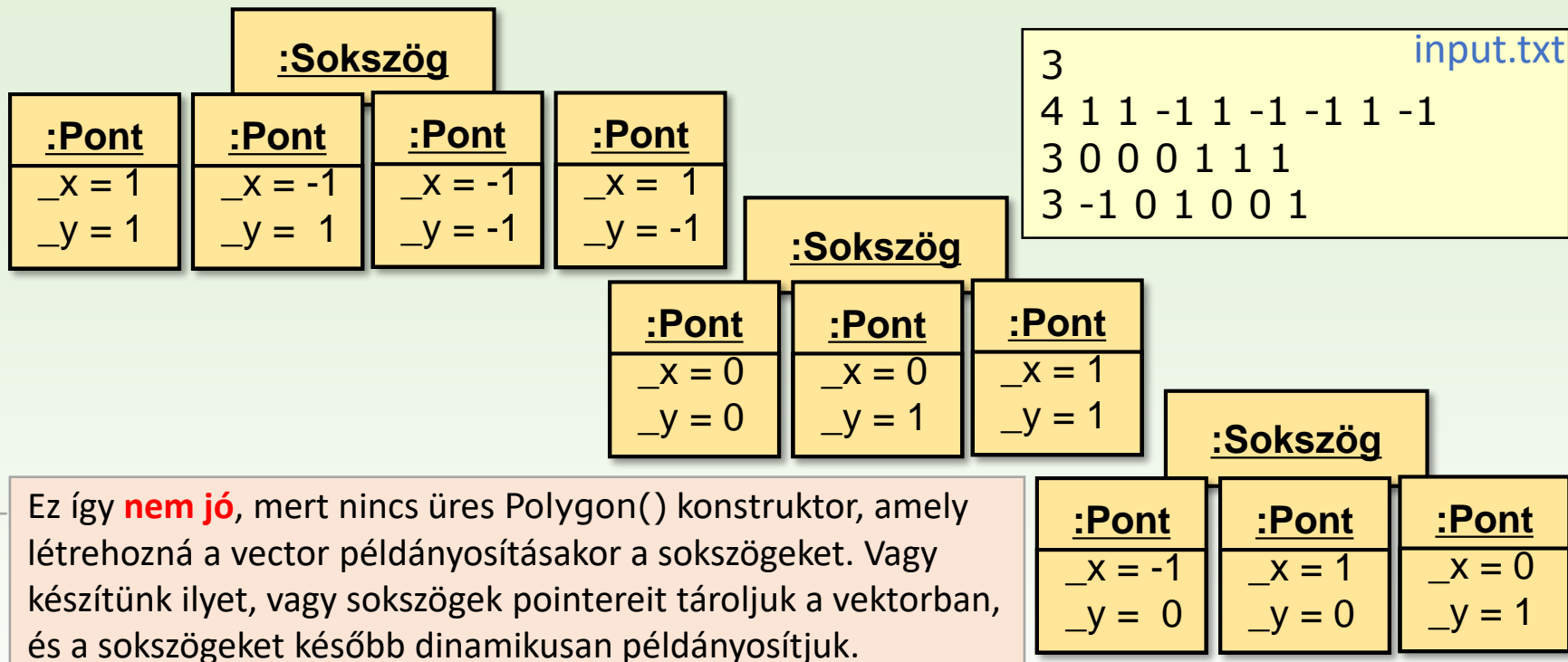
forall (foreach) ciklus az alábbi helyett:

```
for(unsigned int i=0; i<_vertices.size(); ++i){
```

```
    center = center + _vertices[i];
```

```
}
```

A feladat felpopulálása



```
cout << "file name: "; string fn; cin >> fn;
ifstream inp(fn.c_str());
if(inp.fail()) { cout << "Wrong file name!\n"; exit(1); }
```

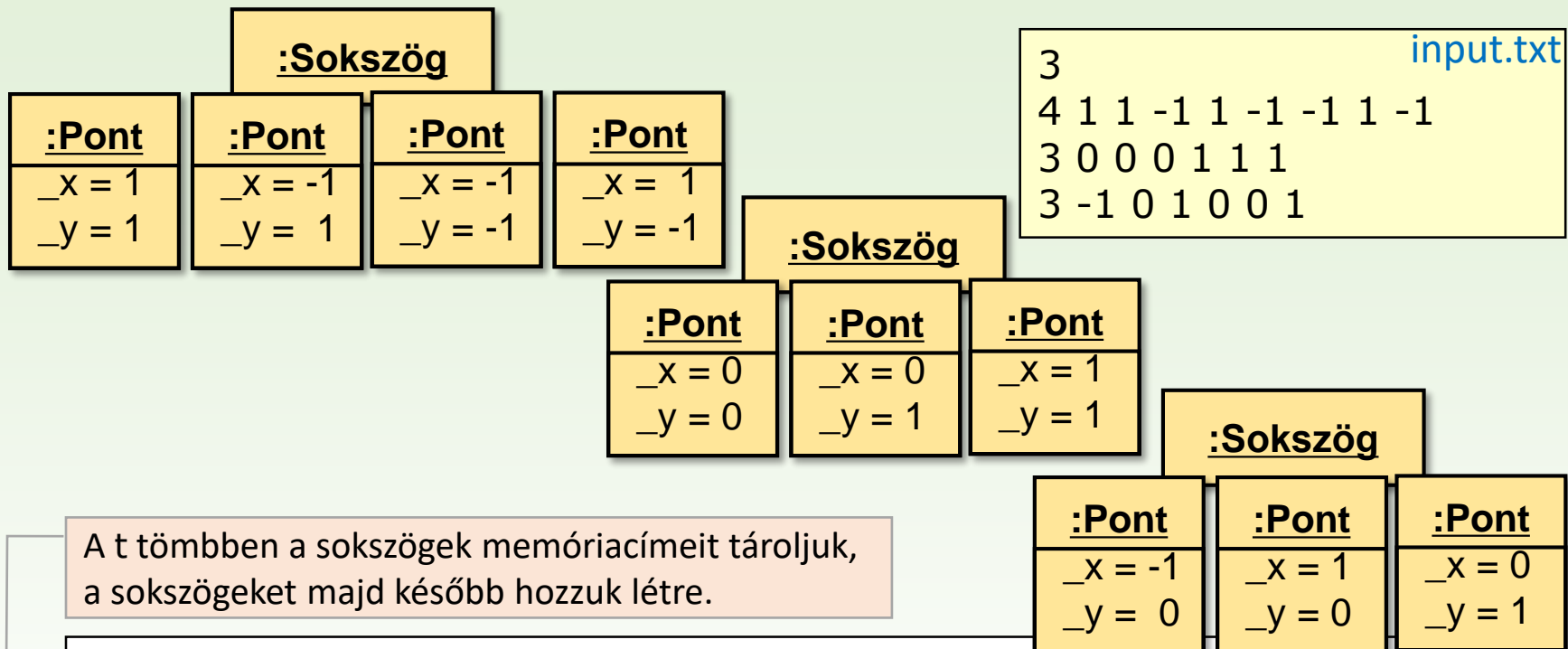
```
#include <cstdlib>
```

```
int n; inp >> n;
vector<Polygon> t(n);
for( unsigned int i=0; i<n; ++i ) t[i] = set(inp);
```

C stílusú karakterlánc

beállítaná az i-dik sokszöget a fájl következő sorának adatai alapján

A feladat felpopulálása újra



A t tömbben a sokszögek memóriacímeit tároljuk, a sokszögeket majd később hozzuk létre.

```

cout << "file name: "; string fn; cin >> fn;
ifstream inp(fn.c_str());
if(inp.fail()) { cout << "Wrong file name!\n"; exit(1); }
    
```

```

unsigned int n; inp >> n;
vector<Polygon*> t(n);
for( unsigned int i=0; i<n; ++i ) t[i] = create(inp);
    
```

Dinamikus tárfoglalással hoz létre egy sokszöget (**new**) a fájl következő sorának adatai alapján.

```

for( Polygon* p : t ) delete p;
    
```

A t vektorban tárolt memóriacímenek található sokszögeket a végén meg kell megszüntetnünk.

Sokszög dinamikus példányosítása

```
class Polygon {  
public:  
    Polygon(int n) : _vertices(n)  
    {  
        if (n < 3) throw FEW_VERTICES;  
    }  
    ...  
private:  
    vector<Point> _vertices;  
};
```

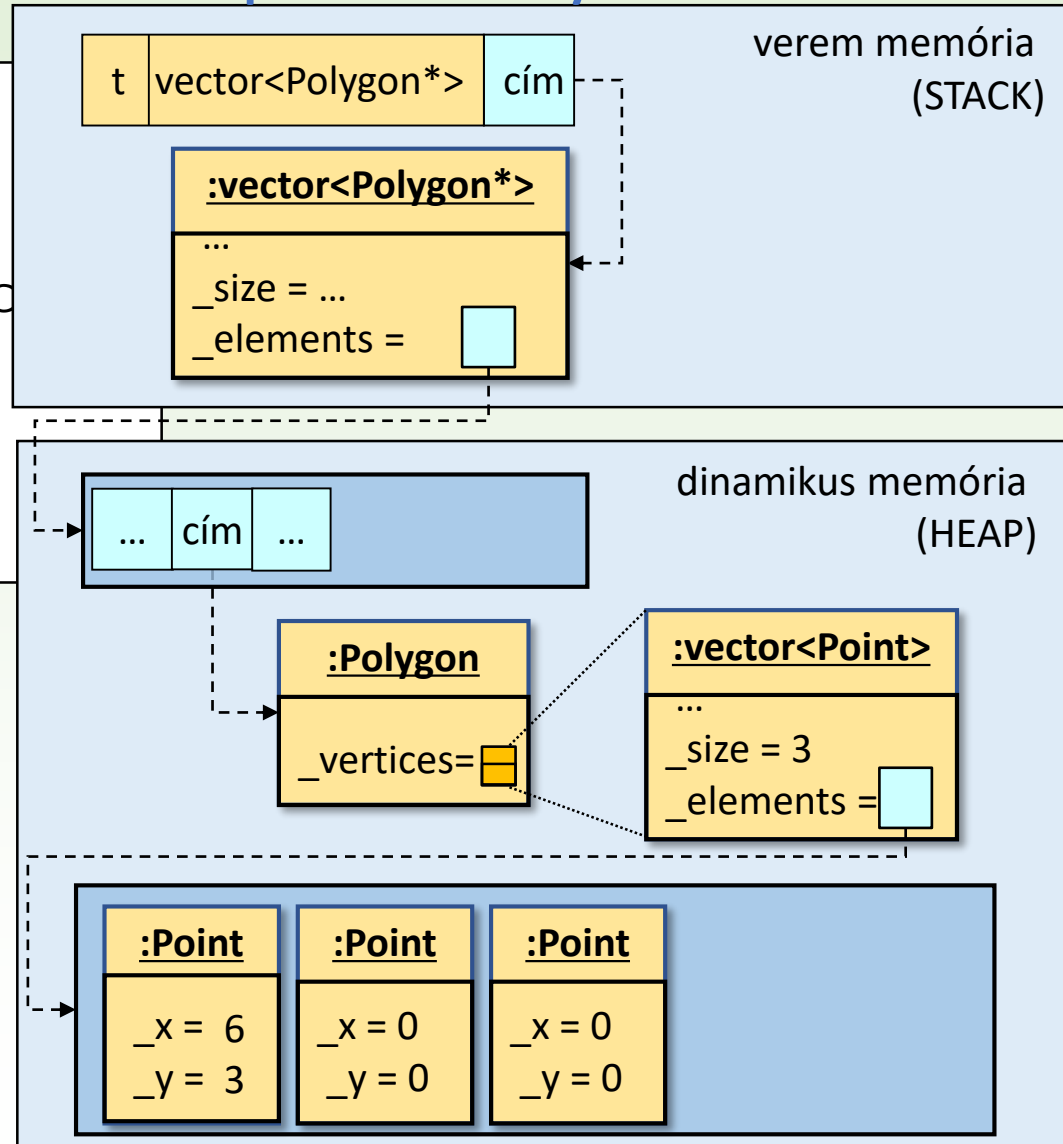
```
vector<Polygon*> t(n);
```

```
t[i] = new Polygon(3);
```

ha nem lenne privát

```
t[i]-> _vertices[0].setPoint(6,3);
```

```
Point c = t[i]->center();
```



Sokszög létrehozása

```
Polygon* create(ifstream &inp) {  
    Polygon *p;  
    try{  
        int sides;  
        inp >> sides;  
        p = new Polygon(sides);  
        for(int i=0; i < sides; ++i){  
            int x, y;  
            inp >> x >> y;  
            p->_vertices[i].setPoint(x,y);  
        }  
    } catch(Polygon::Errors e){  
        if(e==Polygon::FEW_VERTICES) cout << " ... ";  
    }  
    return p;  
}
```

input.txt

```
3  
4 1 1 -1 1 -1 -1 1 -1  
3 0 0 0 1 1 1  
3 -1 0 1 0 0 1
```

a create() nem a Polygon osztály metódusa, így nem férünk hozzá a privát _vertices adattaghoz

Gyártó függvény

```
Polygon* Polygon::create(ifstream &inp)
```

```
{  
    Polygon *p;  
    try{  
        int sides;  
        inp >> sides;  
        p = new Polygon(sides);  
        for(int i=0; i < sides; ++i){  
            int x, y;  
            inp >> x >> y;  
            p->_vertices[i].setPoint(x,y)  
        }  
    }catch(Polygon::Error e){  
        if( e==Polygon::Error::InvalidSides){  
            return 0;  
        }  
    }  
    return p;  
}
```

A Polygon osztály metódusa kellene, hogy legyen, de nem lehet Polygon-ra meghívni, hiszen éppen neki kell a Polygon-t létrehoznia.

```
class Polygon {  
public:  
    enum Errors { ... };  
    Polygon(int n);  
    ...  
    static Polygon* create(std::ifstream &inp);  
private:  
    vector<Point*> _vertices;  
};
```

legyen osztályszintű metódus

osztályszintű metódus hívása

```
t[i] = Polygon::create(inp);
```

Főprogram

Tömbbeli sokszögek eltolása ugyanazon mértékkel, majd ezen sokszögek súlypontjainak kiszámolása.

$A : t : \text{Sokszög}^n, mp : \text{Pont}, cout : \text{Pont}^n$

$Ef : t = t_0 \wedge mp = mp_0$

elemek sorozatba fűzésének jele:
 $\forall i \in [1 .. n] : t[i].\text{eltol}(mp)$

$Uf : mp = mp_0 \wedge t = \bigoplus_{i=1}^n < t_0[i].\text{eltol}(mp) > \wedge$

$\wedge cout = \bigoplus_{i=1}^n < t[i].\text{súlypont}() >$

$cout := <>$

$i = 1 .. n$

$t[i].\text{eltol}(mp)$

$cout := cout \oplus < t[i].\text{súlypont}() >$

Két összegzés (másolás):

$i \in [m..n]$	\sim	$i \in [1 .. n]$	$i \in [1 .. n]$
s	\sim	t	$cout$
$f(i)$	\sim	$< t_0[i].\text{eltol}(mp) >$	$< t[i].\text{súlypont}() >$
$H, +, 0$	\sim	$\text{Sokszög}^*, \oplus, <>$	$\text{Pont}^*, \oplus, <>$

```

for( Polygon *p : t ){
    p->move(Point(20,20));
    p->write();
    Point sp = p->center();
    cout << "(" << sp._x << "," << sp._y << ")\n";
}
    
```

Típusorientált megoldás

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <vector>
#include "polygon.h"
#include "point.h"
using namespace std;
```

```
int main()
{
```

```
    cout << "file name: "; string fn; cin >> fn;
    ifstream inp(fn.c_str());
    if(inp.fail()) { cout << "Wrong file name!\n"; exit(1);}
    int n; inp >> n;
    vector<Polygon*> t(n);
```

populálás

```
    for (unsigned int i=0; i<n; ++i ) t[i] = Polygon::create(inp);
```

```
    for ( Polygon* p : t ){
        P->move(Point(20,20)); p->write();
        Point sp = p->center();
        cout << "(" << sp._x << "," << sp._y << ")\n";
```

számolás

```
    }
    for ( Polygon* p : t ) delete p;
    return 0;
```

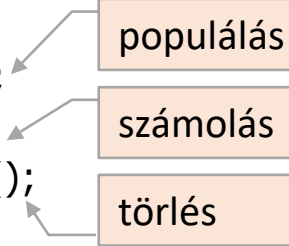
törlés

```
}
```

Objektum-orientált megoldás

```
int main(){  
    Application a;  
    a.run();  
    return 0;  
}
```

```
class Application{  
    public:  
        Application();  
        void run();  
        ~Application();  
    private:  
        std::vector<Polygon*> t;  
};
```



```
Application::Application(){  
    cout << "file name: "; string fn; cin >> fn;  
    ifstream inp(fn.c_str());  
    if(inp.fail()) {  
        cout << "Wrong file name!\n"; exit(1);  
    }  
    unsigned int n; inp >> n;  
    t.resize(n);  
    for(unsigned int i=0; i<n; ++i)  
        t[i] = Polygon::create(inp);  
}
```

```
void Application::run(){  
    for ( Polygon* p : t ){  
        p->move(Point(20,20)); p->write();  
        Point sp = p->center();  
        cout << "(" << sp._x << ", "  
            << sp._y << ")\n";  
    }  
}
```

```
Application::~~Application(){  
    for ( Polygon* p : t ) delete p;  
}
```

Menüvezérelt objektum-orientált megoldás

```
int main()
{
    Menu a;
    a.run();
    return 0;
}
```

```
class Menu{
public:
    Menu(){s = nullptr;}
    void run();
    ~Menu(){ if(s!=nullptr) delete s;}
private:
    Polygon* s;

    void menuWrite();
    void case1();
    void case2();
    void case3();
    void case4();
};
```

egy sokszöget létrehozó,
kiíró, eltoló, súlypontját
kiszámoló metódusok

```
void Menu::run()
{
    int v = 0;
    do{
        menuWrite();
        cin >> v; // ellenőrzés!
        switch(v){
            case 1: case1(); break;
            case 2: case2(); break;
            case 3: case3(); break;
            case 4: case4(); break;
        }
    }while(v != 0);
}
```

```
void Menu::menuWrite(){
    cout << "0 - exit\n";
    cout << "1 - create\n";
    cout << "2 - write\n";
    cout << "3 - move\n";
    cout << "4 - center\n";
}
```


Menüpontok

input1.txt

4 1 1 -1 1 -1 -1 -1 1

input2.txt

3 0 0 -1 0 0 -1

```
void Menu::case1(){  
    if(s!=nullptr) delete s;  
    cout << "file name: "; string fn; cin>> fn;  
    ifstream inp(fn.c_str());  
    if(inp.fail()){ cout << "Wrong file name!\n"; return;}  
    s = Polygon::create(inp);  
}
```

```
void Menu::case2(){  
    if(s==nullptr){ cout << "There is no polygon!\n"; return;}  
    s->write();  
}
```

```
void Menu::case3(){  
    if(s==nullptr){ cout << "There is no polygon!\n"; return;}  
    s->move(Point(20,20));  
}
```

```
void Menu::case4(){  
    if(s==nullptr){ cout << "There is no polygon!\n"; return;}  
    Point sp = s->center();  
    cout << "(" << sp._x << "," << sp._y << ")\n";  
}
```