

Objektumok példányosítása

Gregorics Tibor

gt@inf.elte.hu

<http://people.inf.elte.hu/gt/oep>

Objektum különböző nézőpontokból

Modellezési szempontból

- ❑ Az objektum a megoldandó problémának egy önálló **egyedként azonosítható** része.
- ❑ Az objektum **elrejt**i a felelősségi köréhez tartozó adatokat: azokat kizárólag a metódusai révén kezeljük (olvassuk, módosítjuk).
- ❑ Az objektumnak van **életciklusa** : objektum létrejöttével kezdődik, és megszűnésével fejeződik be.

Programnyelvi megközelítés

- ❑ Az objektum az a **memória-foglalás**, ahol a hozzátartozó adatokat tároljuk.
- ❑ Az objektum adattagjainak és metódusainak) **láthatósági köre** szabályozható, de az objektum metódusai mindig elérik azokat.
- ❑ Egy objektumnak a **konstruktor**a foglal memóriát (példányosítás), és a **destruktor**a törli.

Objektum UML jelölése

□ Egy objektumot meghatároz

- az **osztálya**, amely az ugyanolyan adattagokkal és metódusokkal rendelkező objektumokat írja le.
- a **neve** (amit nem kötelező megadni),
- az **állapota** (amit az adattagjainak értékei jelölnek ki).

<objekturnév>:<osztálynév>

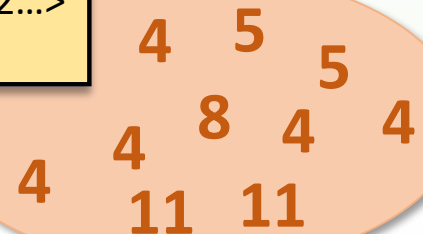
<adattagnév> = <érték>

...

b:Zsák

vec : <0,0,0,0,5,2,0,0,1,0,0,2...>

max : 4



4 5 5 4 4 8 4 4 11 11

Feladat

Töltsünk fel egy tömböt különféle sokszögekkel, és mindegyiket toljuk el ugyanazon irányba és mértékkel, majd számoljuk ki az így nyert sokszögek súlypontjait. A csúcspontok és súlypontok koordinátái, sőt az eltolást leíró helyvektor végpontjának koordinátái is legyenek egész számok.

Objektumok:

- **sokszögek**
- **síkbeli pontok** (sokszögek csúcsai, súlypontok, eltolás helyvektorának végpontja)
- **tömb** a sokszögek tárolásához

Single responsibility

O
L
I
D

Objektumok felelősségi köre:

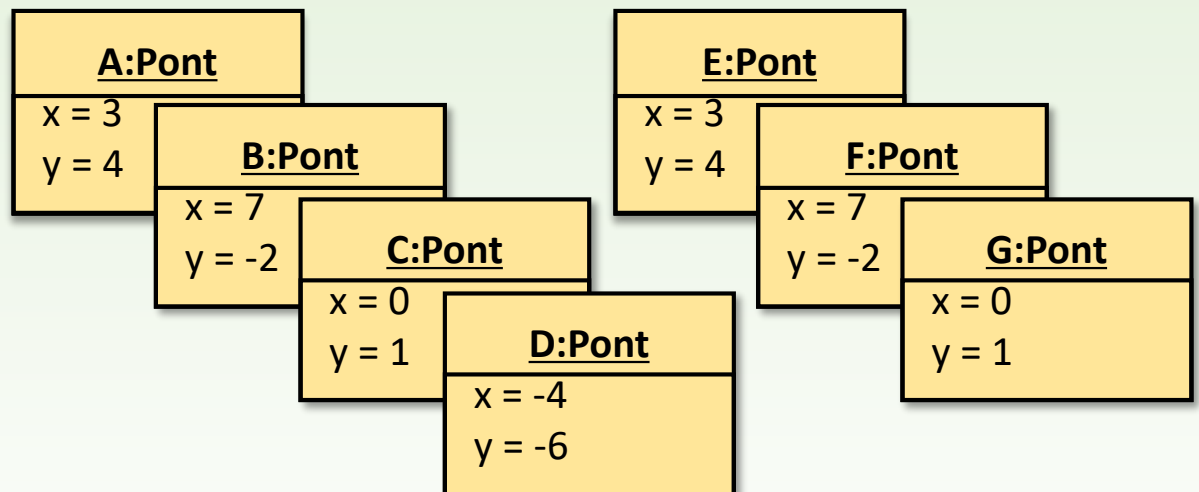
- sokszög: **eltolása, súlypontjának kiszámítása, oldalszámának megadása, adott csúcspontjának lekérdezése és módosítása**
- síkbeli pont: **eltolása, koordinátáinak lekérdezése és módosítása**
- tömb: **adott indexű elem elérése, hosszának lekérdezése**

Objektumok egy populációja

Pontok osztálya:

Pont
x : int
y : int
eltol()

Pont objektumok:



Sokszögek osztálya:

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

Sokszög objektumok:

Négy:Sokszög
csúcsok = < A, B, C, D >

Három:Sokszög
csúcsok = < E, F, G >

Osztályleírás részletezettsége

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

<osztálynév>

<osztálynév>

<metódusnév>()

<osztálynév>

- <adattagnév> : <típusnév>

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

Osztályleírás kiegészítései

- ❑ Az osztálynak a viselkedési tulajdonságát a `<<...>>` jelzés között írhatjuk le (pl. `<<interface>>` , `<<enumeration>>`).
- ❑ Az osztály általánosítását (osztály-sablon) szolgálja a később megadandó **paraméterek** alkalmazása (pl. típust helyettesítő paraméterek).
- ❑ Az adattagokhoz **kezdőértéket** rendelhetünk (ezt a konstruktor állítja majd be), és a `{...}` jelzésben **megszorításokat** (típus invariáns) tehetünk.
- ❑ A metódusok működésére ad **megszorításokat** például `{query}`, amely az adattagokat nem módosító műveletet jelzi. Az adattagok értékét lekérdező műveletek csoportját a `<<getter>>`, a felülíró műveletek csoportját a `<<setter>>` vezeti be.

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

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

Pont fogalmának modellezése

Elemzés szintje

Pont
x : int
y : int
eltol()

Tervezés szintje

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

Tervezési döntések:

- legyenek az adattagok **publikusak**
- az **eltolás paramétere** legyen az eltolás irányát és mértékét megadó helyvektor végpontja
- az eltolás ne változtassa meg azt a pontot, amire meghívják (**query**), hanem újat hozzon létre: **c := a.eltol(b)**
// c az a-hoz képest b-vel eltol pont

```
c : Pont
c.x := x + mp.x
c.y := y + mp.y
return c
```

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

```
return Pont( x / n, y / n)
```

Megvalósítás szintje

Pont
+ x : int
+ y : int
+ Pont()
+ Pont(a:int, b:int)
<<setter>>
+ setPont(a:int, b:int)
+ operator<<(...) : void {query}
+ eltol(mp:Pont) : Pont {query}
+ operator+(mp:Pont): Pont {query}
+ operator/(n:int) : Pont {query}

legyen kétféle konstruktor

adattagok módosítása

kiíró metódus

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

új művelet a súlypont kiszámításához: s := s / n

x, y := 0, 0

x, y := a, b

Pont osztály C++ kódja

```
class Point
```

```
{
```

```
public:
```

```
    int _x, _y;
```

```
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 n) const
    { return Point(_x / n, _y / n ); }
```

```
};
```

paraméterváltozók default értéke:

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

konstans referencia paraméterváltozó

query

operátor felüldefiniálás:

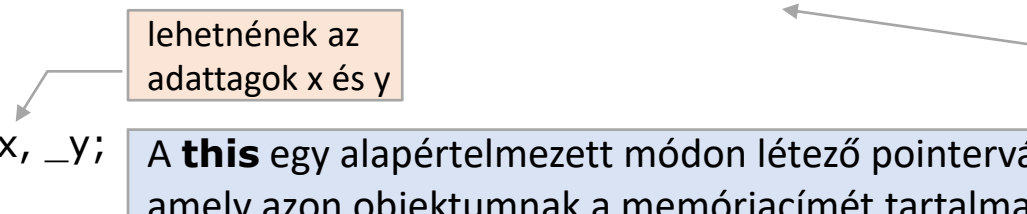
p + q kifejezés meghívja a p objektumra az operator+()-t, a q paraméterre az mp hivatkozik, és egy pontot ad vissza.

A kiíró operátor nem a Point osztály metódusa, hisz cout<<... formában kell a cout-ra meghívni, nem p<<... alakkal. Ezért külső operátorként vezetjük be.

```
std::ostream& operator<<(std::ostream &o, const Point &p)
{
    o << "(" << p._x << "," << p._y << " ";
    return o;
}
```

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;  
};
```



lehetnének az
adattagok x és y

A **this** egy alapértelmezett módon létező pointerváltozó, amely azon objektumnak a memóriacímét tartalmazza (arra mutat), amelyik objektumra a metódust meghívták: Pl. a `p.setPoint(3, -2)` hívás esetén a `this` a `p` címe.

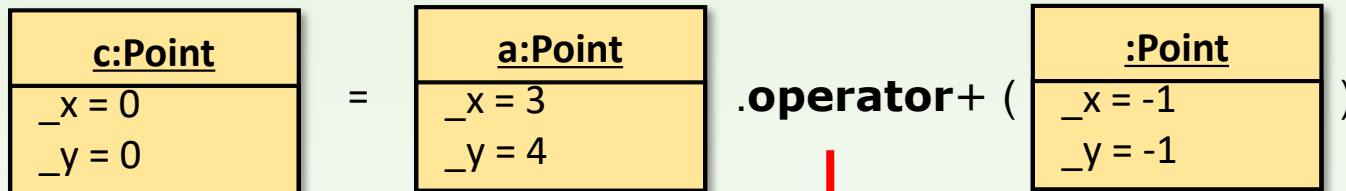
3

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

Pont objektumok példányosítása

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

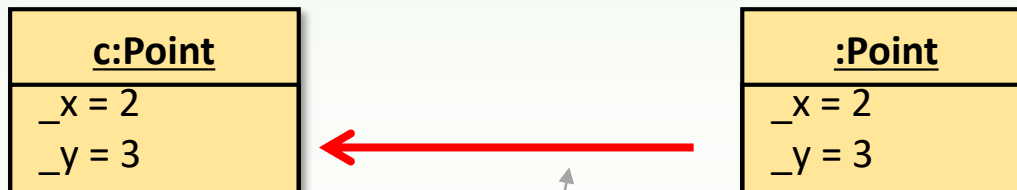
`operator+()` hívása előtt:



Erre a pontra hivatkozik a **operator+()** metódus mp paraméterváltozója.

`operator+()` hívása közben:

`operator+()` hívása után:



Ez az **operator+()**-ban létrehozott pont.

Minden objektum rendelkezik **értékadás operátorral**, amely – ha nem változtatjuk meg – lemásolja az adattagok értékét.

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

Sokszög fogalmának modellezése

Elemzés szintje

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

Tervezési döntések:

- adattag legyen **privát**
- az eltolás azt a sokszög objektumot, azaz annak csúcsait tolja el, amire meghívják (**nem query**)

Tervezés szintje

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

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

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

Megvalósítás szintje

Sokszög
- csúcsok : vector<Pont>
+ Sokszoeg(n:int)
<<getter>>
+ oldalszam() : int {query}
+ eltol(mp:Pont) : void
+ súlypont() : Pont {query}
+ operator<<(...) : void {query}

a csúcsok száma

csúcsok eltolása

kiíró metódus

csúcsok eltolása

Sokszög osztály

Megvalósítás szintje

Sokszög	
- csúcsok : vector<Pont>	
+ Sokszög(n:int)	○
<<getter>>	
+ oldalszám() : int {query}	○
+ eltol(mp:Pont) : void	○
+ súlypont() : Pont {query}	○
+ operator<<(o:ostream) : void {query}	○

A forall (foreach) olyan ciklus, amelyik bejárja (de nem változtatja meg) egy gyűjtemény elemeit.

```
for i=1 .. csúcsok.hossz() loop
  ... csúcsok[i] ...
endloop
```

hibaellenőrzés

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

```
return csúcsok.size()
```

```
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 / csúcsok.size()
```

```
o << ">"
forall csúcs in csúcsok loop
  o << csúcs
endloop
o << ">"
```

Sokszög osztály C++ kódja

```
class Polygon
{
private:
    std::vector<Point> _vertices;
public:
    enum Errors{FEW_VERTICES};

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

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

    void move(const Point &mp);

    Point center() const;

    friend std::ostream& operator<<(std::ostream &o, const Polygon &p);
};
```

hibaesetek (kivételek) definiálása

A hibát csak jelezzük (kivétel dobása), de a hiba kezelése nem itt történik.

tesztelésnél:
CHECK_THROWS(Polygon())

olyan külső operátor, amelyik hozzáfér a privát tagokhoz

Sokszög metódusainak C++ 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(const Point& vertex : _vertices) { center = center + vertex; }  
    return center / sides();  
}
```

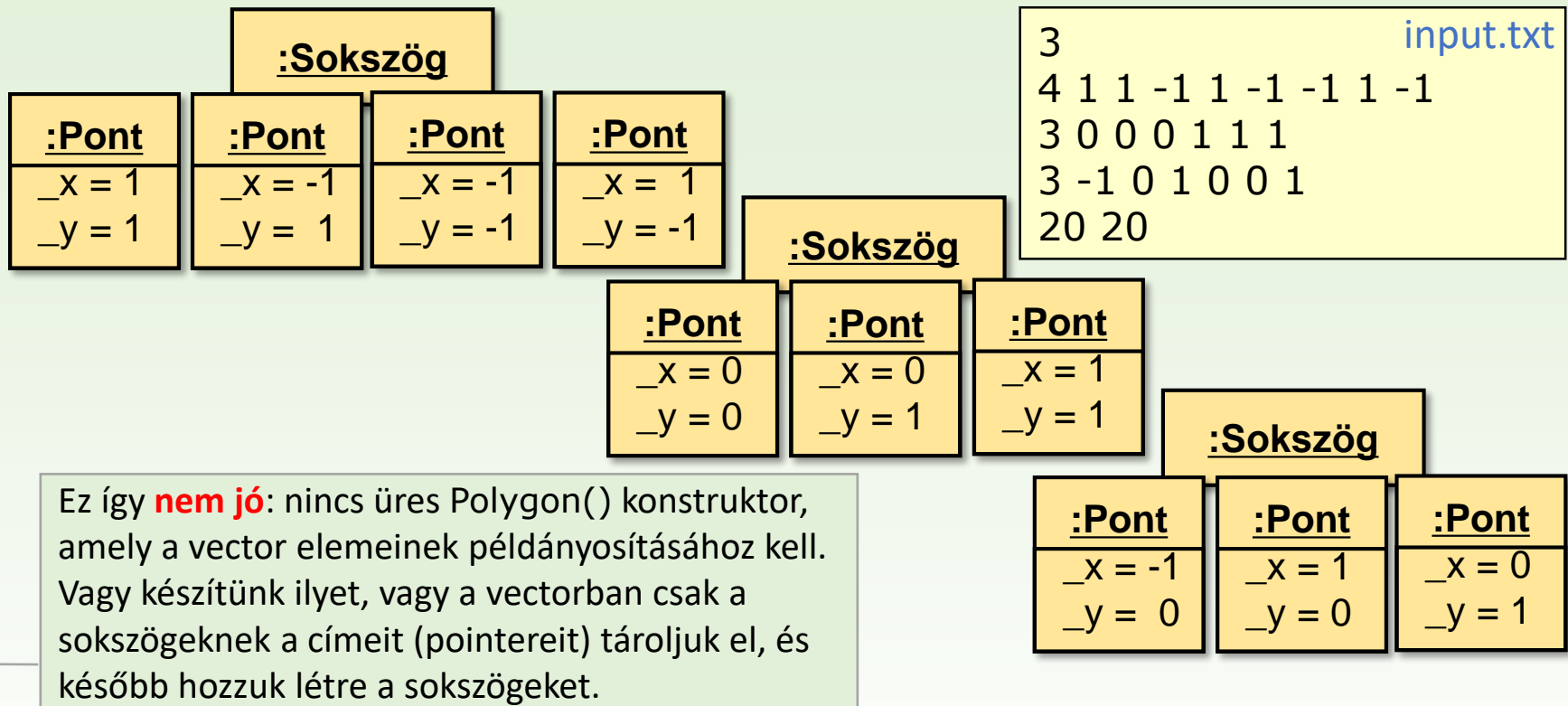
a **forall** (foreach) ciklus C++ kódja

```
std::ostream& operator<<(std::ostream &o, const Polygon &p)
```

```
{  
    o << "<";  
    for( const Point& vertex : p._vertices ) { o << vertex; }  
    o << ">";  
    return o;  
}
```

a friend kapcsolat miatt
hivatkozhat a privát adattagra

A feladat felpopulálása



```
ifstream inp("input.txt");
```

```
...
```

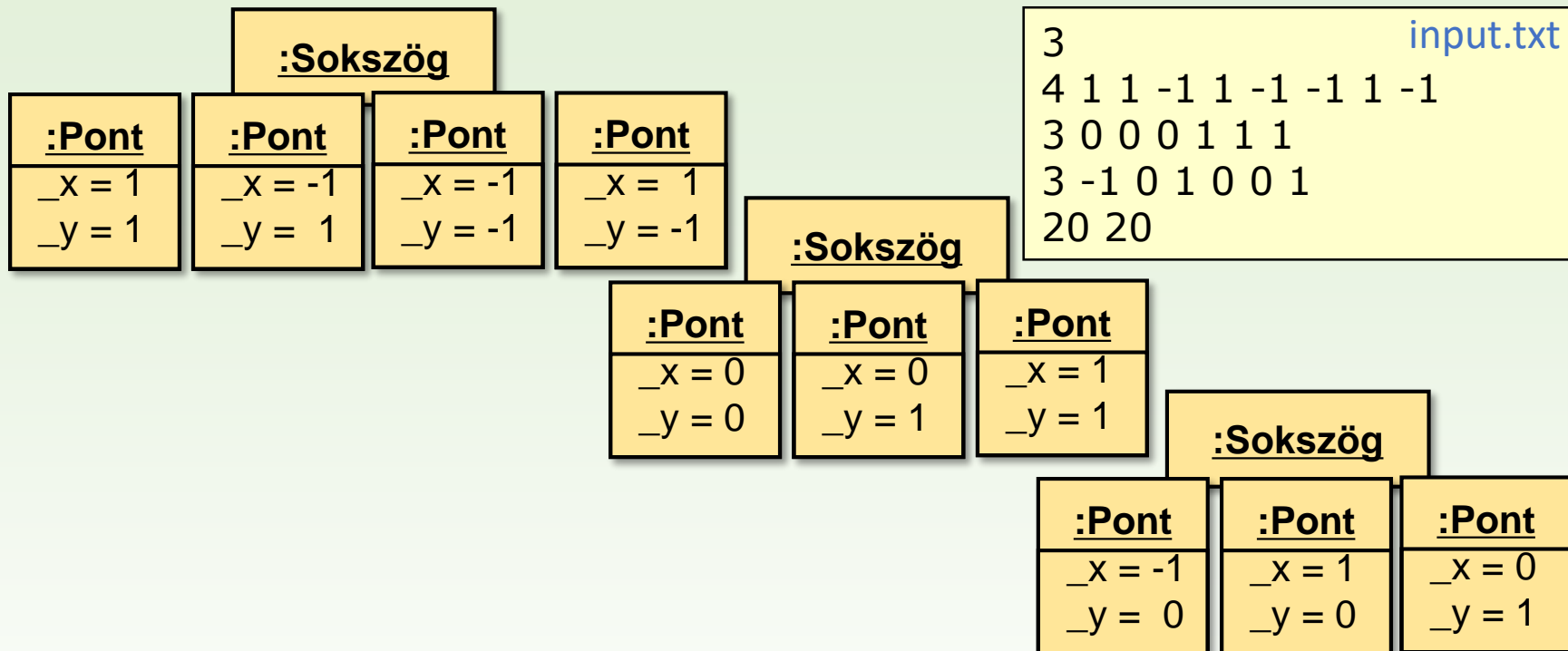
```
int n; inp >> n;
```

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

```
for( int i=0; i<n; ++i ) { t[i].set(inp); }
```

Felparaméterezi az i-dik sokszöget a fájl soron következő sorának adatai alapján.

A feladat felpopulálása újra



```
ifstream inp("input.txt");
```

```
...
```

```
int n; inp >> n;
```

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

```
for( int i=0; i<n; ++i ) { t[i] = create(inp); }
```

Dinamikus példányosítással hoz létre egy sokszöget a fájl következő sorának adatai alapján, majd visszaadja annak címét.

Sokszög dinamikus példányosítása

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

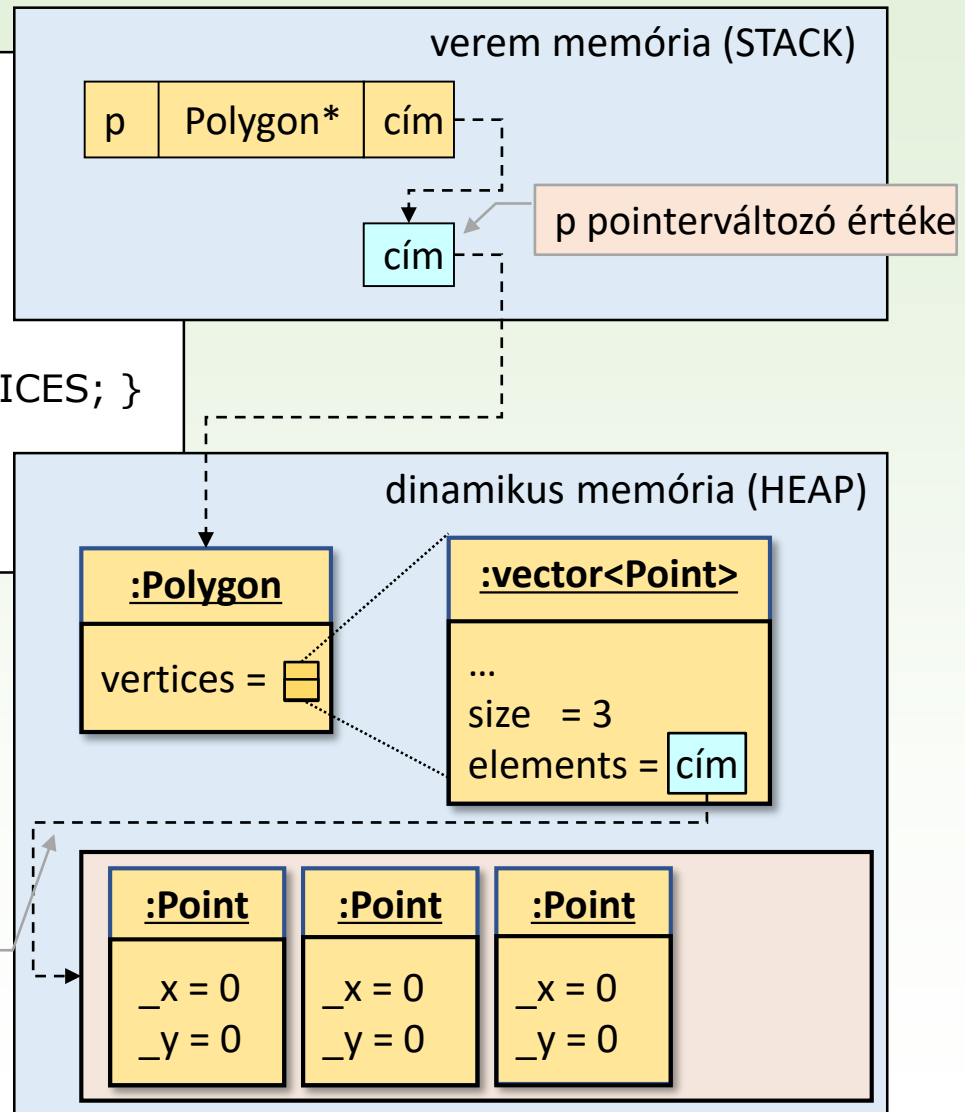
```
Polygon* p;
```

```
p = new Polygon(3);
```

egy három oldalú sokszög
dinamikus tárfoglalása

ezeket a memóiafoglalásokat
nekünk kell majd szüntetnünk

```
delete p;
```



Sokszög létrehozása

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
20 20
```

```
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 << "cannot create the polygon.\n";
        }
    }
    return p;
}
```

hibák figyelése

létrehozza a sokszöget

beállítja a sokszög csúcsainak koordinátáit

Mivel a create() nem a metódusa a Polygon osztálynak, így nem fér hozzá annak privát _vertices adattagjához.

hibakezelés

Objektumot gyártó függvény

```
Polygon* Polygon::create(istream &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_VE
```

```
        return p;
```

```
}
```

Legyen a create() a Polygon osztály metódusa.
De melyik sokszög objektumra kellene meghívni ezt?
Éppen neki kellene létrehoznia egy ilyen objektumot.

```
class Polygon {
```

```
public:
```

```
    enum Errors { ... };
```

```
    Polygon(int n);
```

```
    ...
```

```
    static Polygon* create(std::istream &inp);
```

```
private:
```

```
    vector<Point*> _vertices;
```

```
};
```

Legyen **osztálysintű** a create():
ne egy sokszög objektumhoz,
hanem a sokszögek osztályához
tartozzon.

osztálysintű metódus hívása

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

Főprogram

Tömbbeli sokszögek eltolása ugyanazon irányban és mértékkel, majd súlypontjaiknak kiszámolása.

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

$Ef : t = t' \wedge mp = mp'$

sorozatok összekapcsolásának jele

$Uf : mp = mp' \wedge t = \bigoplus_{i=1}^n \langle \text{eltol}(t'[i], mp) \rangle \wedge r = \bigoplus_{i=1}^n \langle \text{súlypont}(t[i]) \rangle$

Két összegzés (összekapcsolás):

$i \in [m..n] \sim i \in [1..n]$

$f(i) \sim \langle \text{eltol}(t'[i], mp) \rangle$

$s \sim t$

$H, +, 0 \sim \text{Sokszög}^*, \oplus, \langle \rangle$

a két összegzés közös ciklusba vonható össze

$i \in [1..n]$

$\langle \text{súlypont}(t[i]) \rangle$

r

$\text{Pont}^*, \oplus, \langle \rangle$

az összekapcsolás helyett elég a t elemeit egyenként megváltoztatni: $t[i] := \text{eltol}(t[i], mp)$

$r := \langle \rangle$

$i = 1..n$

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

$t[i] := \text{eltol}(t[i], mp)$

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

$r := r \oplus \text{súlypont}(t[i])$

```
for( Polygon *p : t ) {  
    p->move(mp); cout << *p << endl;  
    cout << p->center() << endl;  
}
```

az r feltöltése helyett a súlypontokat közvetlenül kiírjuk

Típus-orientált megoldás

```
#include "polygon.h"  
#include "point.h"  
using namespace std;
```

```
#include <iostream>  
#include <fstream>  
#include <vector>
```

```
int main()  
{
```

```
    cout << "file name: "; string fn; cin >> fn;  
    ifstream inp(fn);  
    if(inp.fail()) { cout << "File open error\n"; return 1; }  
    int n; inp >> n;  
    vector<Polygon*> t(n);  
    for ( int i=0; i<n; ++i ) t[i] = Polygon::create(inp);  
    int x, y; inp >> x >> y;  
    Point mp(x, y);
```

```
    for ( Polygon* p : t ) {  
        p->move(mp); cout << *p << endl;  
        cout << p->center() << endl;  
    }
```

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

```
}
```

populálás

számolás

törlés

megszünteti a dinamikus
memóriában tett helyfoglalásainkat

Objektum-orientált megoldás

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

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

populálás

számolás

törlés

```
Application::Application(){  
    cout << "file name: "; string fn; cin >> fn;  
    ifstream inp(fn);  
    if(inp.fail()) { cout << "File open error\n"; exit(1);}  
    int n; inp >> n;  
    t.resize(n);  
    for( int i=0; i<n; ++i) t[i] = Polygon::create(inp);  
    int x, y; inp >> x >> y;  
    mp.setPoint(x, y);  
}
```

#include <cstdlib>

```
void Application::run(){  
    for ( Polygon* p : t ) {  
        p->move(mp); cout << *p << endl;  
        cout << p->center() << endl;  
    }  
}
```

A destruktorkor akkor hívódik meg,
amikor az objektum megszűnik.

```
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);  
    if(inp.fail()) { cout << "File open error\n"; return;}  
    s = Polygon::create(inp);  
}
```

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

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

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