PROGRAMOZÁS

Gömb-pont modellezés

Gregorics Tibor

http://people.inf.elte.hu/gt/prog

Feladat



Hány UFO tartózkodik az űrállomás közelében?

Programterv

$$A = (v:UFO^n, s:SpaceStation, c:N)$$

$$Ef = (v = v' \land s = s')$$

$$Uf = (v = v' \land s = s' \land c = \sum_{i=1}^{n} 1$$

v[i] az s közelében van

$$c := 0$$

$$i = 1 ... n$$

v[i] az s közelében van

$$c = c + 1$$



$$A = (v:Point^n, s:Sphere, c:\mathbb{N})$$

$$Ef = (v = v' \land s = s')$$

$$Uf = (v = v' \land s = s' \land c = \sum_{i=1}^{n} 1$$

 $v[i] \in S$



Fő program

```
Sphere s;
... // s tulajdonságainak beállítása
vector Point> v(n);
 ... // a v-beli pontok tulajdonságainak beállítása
int c = 0;
for(int i=0: i<n: ++i) {
    if(s.contains(v[i])) ++c;
                                        A C++ számára
                                        ismeretlen típusok
                                         ismeretlen művelettel.
cout << "result: " << c << endl;</pre>
```

Gömb típusa

a típus értékhalmaza:

egy Sphere típusú adat lehetséges értékei

Sphere = $\{h \in \mathbf{2}^{Point} \mid \exists c \in Point \land \exists r \in \mathbb{R} : \forall p \in h : distance(c,p) \leq r \}$

a típus műveletei:

a gömbökkel végezhető művelet

gömbök

Benne van-e egy pont a gömbben:

l:=contains(s, p) s: Sphere, p: Point, l: L

centre (Point) $radius: \mathbb{R}$

 $l:=distance(s.centre, p) \leq s.radius$

Invariáns: radius >= 0.0

a típus értékeinek reprezentációja.

egy értéknek (gömbnek) a számítógép memóriájában történő ábrázolásához szükséges adatok

a típus műveleteinek implementációja:

a művelet működését leíró program, amely az eredeti típus érték (gömb) helyett az azt helyettesítő középponttal és sugárral dolgozik.

Pont típusa

a típus értékhalmaza.

egy Point típusú adat lehetséges értékei

Point = {térbeli pontok}

a típus műveletei:

a pontokkal végezhető művelet

térbeli pontok Két pont távolsága:

$$d$$
:= $distance(p_1, p_2)$

$$p_1, p_2$$
: Point, $d : \mathbb{R}$

$$x, y, z : \mathbb{R}$$

$$d := sqrt((p_1.x-p_2.x)^2 + (p_1.y-p_2.y)^2 + (p_1.z-p_2.z)^2)$$

a típus értékeinek reprezentációja:

egy értéknek (térbeli pontnak) a számítógép memóriájában történő ábrázolásához szükséges adatok a típus műveleteinek implementációja:

a művelet működését leíró program, amely az eredeti típus érték (pont) helyett az azt helyettesítő koordinátákkal dolgozik.

Tipus

Típus-specifikáció

T

 $repr:R \rightarrow T$

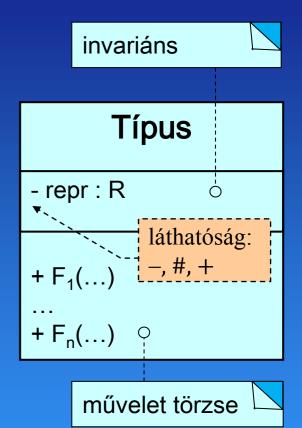
 $inv: R \rightarrow \mathbb{L}$

 F_1 ... F_n olyan feladatok, amelyek állapotterében szerepel a T

 S_1 ...

olyan programok, amelyek állapotterében szerepel az R műveletek

implementáció



Típus-implementáció

a reprezentáció helyessége:

minden *T*-beli érték helyettesíthető *inv*-et kielégítő *R*-beli adattal, és minden *inv*-et kielégítő *R*-beli adat helyettesít *T*-beli értéket.

az implementáció helyessége:

minden feladatot megold egy program abban az értelemben, amilyen értelemben a típusértékeket *inv*-et kielégítő *R*-beli adatokkal lehet helyettesíteni.

Típusok osztálydiagramja

A Sphere nélkülözhetetlen tartozéka a centre : Point privát adattag

Invariáns: radius >= 0.0



- centre : Point
- radius : double
- + contains(p:Point) : bool \circ

- centre

Point

- x : double
- y : double
- z : double
- + distance(p:Point) : double o

return centre.distance(p) ≦ radius

Valójában két paramétere van, mert az objektum orientált hívás módja:

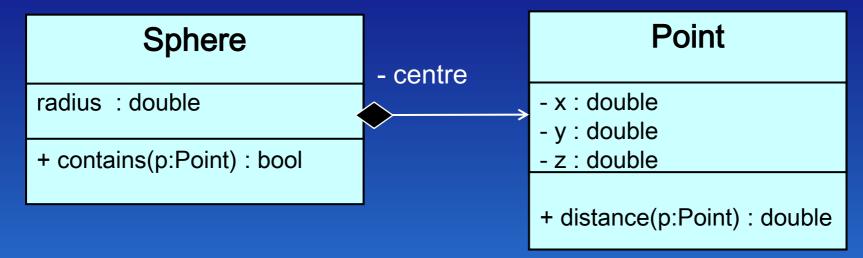
bool I = s.contains(p)

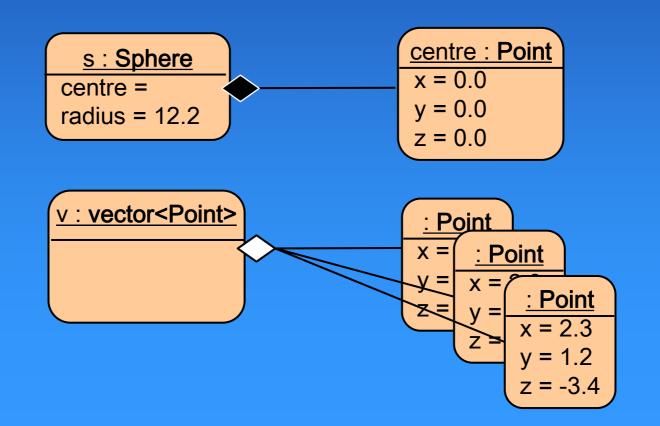
ahol az s:Sphere a kitüntetett (default, this) paraméter, amelynek adattagjaira a metódus törzsében közvetlenül hivatkozhatunk.

return sqrt($(x-p.x)^2+(y-p.y)^2+(z-p.z)^2$)

Az objektum orientált hívás módja: double d = centre.distance(p) ahol centre:Point.

Kompoziciós modell





Point a: Point x =b : Point - x : double **y** = Konstruktor: x = 2.3- y : double z =Point a - z : double y = 1.2Point b(2.3, 1.2, -3.4) + Point() z = -3.4+ Point(double,double,double) + set(double,double,double)

C++ osztály implicit módon mindig rendelkezik egy Pont(){} üres (default) + distance(p:Point) : double konstruktorral, amíg nem definiálunk

```
mást explicit módon.
class Point{
                                                 másképp: Point() {set(0.0,0.0,0.0);}
public:
    Point(): \underline{x}(0.0), \underline{y}(0.0), \underline{z}(0.0) {} Point(double x,double y,double z)
                                                 {set(x,y,z);}
    Point (double x, double y, double z) : x(x), y(y), z(z) {}
    void set (double x, double y, double z) { x = x; y = y; z = z;}
   double distance(const Point &p) const{
        return sqrt(pow( x-p. x,2)+pow( y-p. y,2)+pow( z-p. z,2));
                                                    konstans metódus: nem
private:
   double _x, _y, _z;
                                                    változtatja meg az adattagokat
```

Sphere

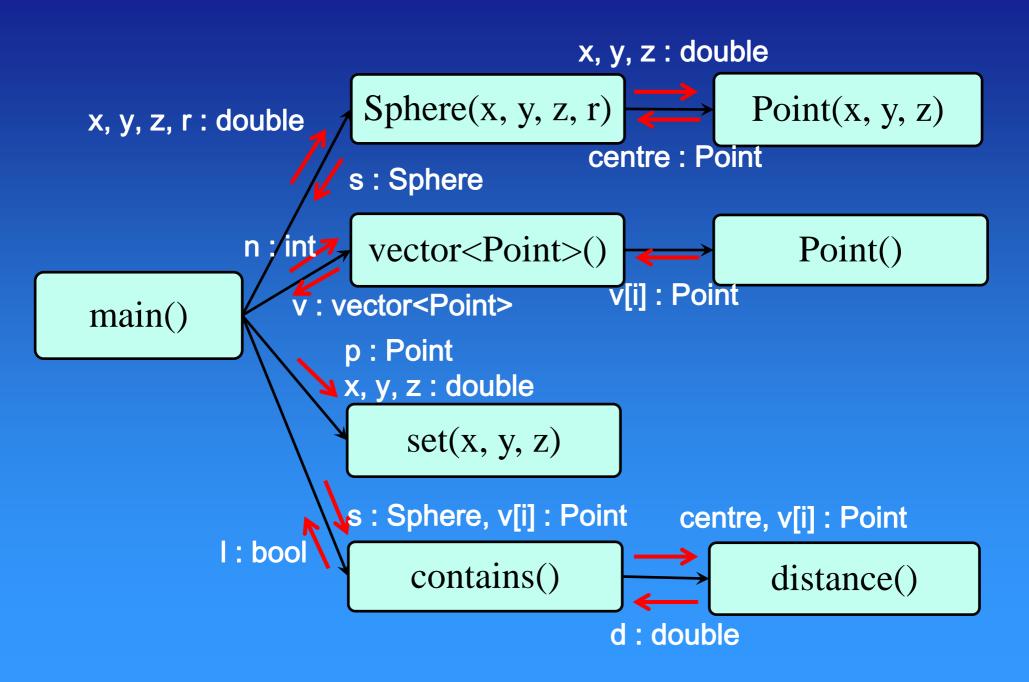
```
centre : Pointradius : double
```

- + Sphere(double,double,double,double)
- + contains(p:Point) : bool

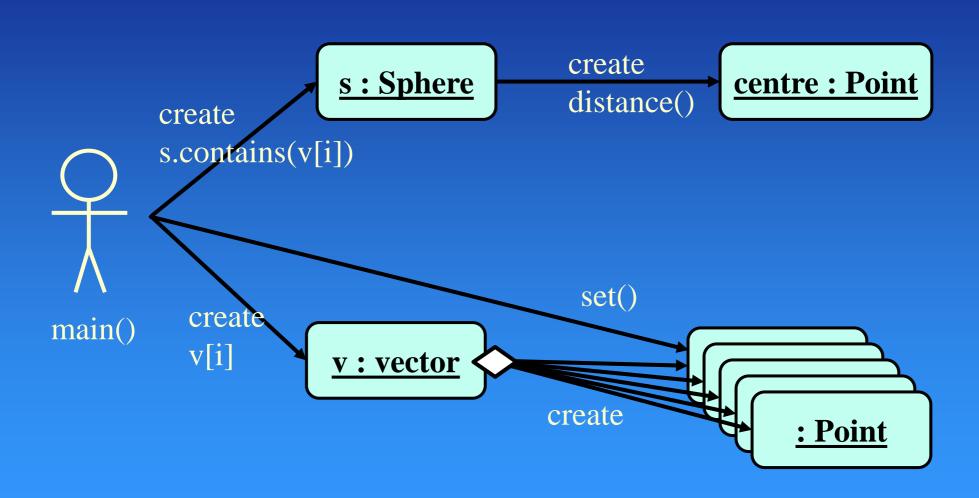
```
class Sphere {
public: beágyazott felsorolás típus
                                              Sphere(...) { _centre.set(x,y,z); ... }
   enum Errors { NEGATIVE RADIUS };
   Sphere (double x, double y, double z, double r) : centre (x, y, z) {
       if(r<0) throw NEGATIVE RADIUS; radius = r;</pre>
                                                          Point metódusait hívják
   bool contains (const Point &p)_const-{
       return centre.distance(p) <= radius;
private:
                            A konstruktor ellenőrzi az invariánst:
   Point centre;
                            hibás input esetén kivételt dob.
   double radius;
};
```

```
int main()
   double x, y, z, r;
    ... // x,y,z,r beolvasása
   try{
       Sphere s(x,y,z,r); Sphere(x,y,z,r) azon belül Point(x,y,z)
   }catch (Sphere::Errors err) {
       if(err==Sphere::NEGATIVE RADIUS) cout << ...;</pre>
                                              kivétel lekezelése
   int n; cin >> n;
   // n ellenőrzése ...
                             Point() n-szer
   vector<Point> v(n);
   for(int i=0; i<n; ++i) {
       cout << i+1 << ". points:" << endl;
       ... // x,y,z beolvasása
       v[i].set(x,y,z);
   int c = 0;
   for (int i=0; i<n; ++i) {</pre>
       if(s.contains(v[i])) ++c;
   cout << "result: " << c << endl;</pre>
   return 0;
```

Modulszerkezet (procedurális szemlélet)



Együttműködési diagram (objektum orientált szemlélet)



Csomag szerkezet

main.cpp

```
#include <iostream>
#include <vector>
#include "sphere.h"
#include "point.h"

using namespace std;

int main() { ... }
```

sphere.h

point.h

```
#ifndef _SPHERE_H
#define _SPHERE_H

#include "point.h"
class Sphere{...};

#endif // _SPHERE_H
```

```
#ifndef _POINT_H
#define _POINT_H

class Point{...};

#endif // _POINT_H
```

Fekete doboz tesztelés vázlata

Számlálás tesztje:

```
intervallum (1...n) eleje: csak a tömb legelső pontja esik bele a gömbbe intervallum (1...n) vége: csak a tömb utolsó pontja esik bele a gömbbe intervallum (1...n) vége: csak a tömb utolsó pontja esik bele a gömbbe intervallum (1...n) túlindexelése: nagyon nagy sugarú gömb n=0 mellett funkció szerint: 0, 1 vagy több pont is legyen a gömbben
```

Osztályok tesztje

- Egyenként teszteljük a metódusokat.
- Külön ügyelünk arra, hogy a metódusok (köztük a konstruktorok) nem sértik-e meg a típus-invariánst (ezt a konstans metódusoknál nen kell vizsgálni).
- Integrációs teszt: a metódusok tetszőleges variációinak végrehajtásait kell vizsgálni.

Egy másik modellje a feladat típusainak

Point	Két pont távolsága: $d:=distance(p_1, p_2)$ $p_1, p_2:$ Point, $d:\mathbb{N}$
$x, y, z : \mathbb{R}$	$d:=sqrt((p_1.x-p_2.x)^2+(p_1.y-p_2.y)^2+(p_1.z-p_2.z)^2)$

Egy gömb legyen olyan, mint egy pont, csak legyen még neki sugara is és legyen egy contains() művelete.

Sphere

Benne van-e egy pont a gömbben:

l:=contains(s, p) s: Sphere, p: Point, l: L

az x, y, z : ℝ -t örökli

 $x, y, z : \mathbb{R}$

 $l := distance(s, p) \leq s.radius$

 $radius: \mathbb{R}$

Invariáns: radius >= 0.0

a distance()-t örökli, ami egy gömbre és egy pontra a gömb középpontja és a pont közötti távolságot adja meg.

Származtatásos modell

Point

- x : double
- y : double
- z : double
- + Point()
- + Point(double,double,double)
- + set(double,double,double): void
- + distance(p:Point) : double

return $sqrt((x-p.x)^2+(y-p.y)^2+(z-p.z)^2)$

A Sphere rendelkezik a Point elemeivel (örökli azokat), bár közvetlenül nem éri el annak privát (-) tagjait. (a konstruktorok nem öröklődnek)

Sphere

Önmagában egy s gömb és egy p pont távolságára az örökölt distance() metódus nem ad jó eredményt, ez csak a contains() céljainak felel meg.

- radius : double O---- Invariáns: radius >= 0.0
- + Sphere(double,double,double,double)
- + contains(p:Point) : bool ○
- + distance(p:Point) : double

return Point::distance(p) ≦ radius

return Point::distance(p) - radius

Point

```
- x : double
- y : double
- z : double
+ Point()
+ Point(double,double,double)
+ set(double,double,double) : void
+ distance(p:Point) : double
```

```
class Point{
public:
   Point():_x(0.0), _y(0.0), _z(0.0) {}
   Point (double x, double y, double z) : x(x), y(y), z(z) {}
   void set (double x, double y, double z) { x = x; y = y; z = z;}
   double distance(const Point &p) const{
      return sqrt(pow(x-p. x,2)+pow(y-p. y,2)+pow(z-p. z,2));
private:
   double _x, _y, _z;
```

Sphere

- radius : double
- + Sphere(double,double,double,double)
- + contains(p:Point) : bool

protected származtatás: az öröklött publikus tagokat védetté teszi, így a distance() metódus is védett lesz, azaz kívülről gömbökre nem használható.

```
protected Point {
class Sphere
public:
                  származtatás
                                         Az ősosztály konstruktora
                                        hívódik meg az öröklött
   enum Errors { NEGATIVE RADIUS };
                                         adattagok beállítására.
   Sphere (double x, double y, double z, double r) : Point (x, y, z) {
       if( r<0) throw NEGATIVE RADIUS; radius = r;</pre>
   bool contains (const Point &p) const {
       return distance(p) <= radius;
private:
   double radius;
```

Ugyanaz a fő program

```
int main()
   double x,y,z,r;
   ... // x,y,z,r beolvasása
   try{
       Sphere s(x,y,z,r);
   catch (Sphere::Errors err) {
       if(err==Sphere::NEGATIVE RADIUS) cout << ...;</pre>
   int n; cin >> n;
   vector<Point> v(n);
   for(int i=0; i<n; ++i) {
       cout << i+1 << ". points:" << endl;
       ... // x,y,z beolvasása
       v[i].set(x,y,z);
   int c = 0;
   for (int i=0; i<n; ++i) {</pre>
       if(s.contains(v[i])) ++c;
   cout << "result: " << c << endl;</pre>
   return 0;
```

Egy újabb modellje a feladat típusainak

Sphere	Két gömb távolsága:
	$d:=distance(s_1, s_2)$ $s_1, s_2:$ Sphere, $d:\mathbb{N}$
	Benne van-e egy pont a gömbben:
	$l:=contains(s_1, s_2)$ $s_1, s_2:$ Sphere, $l:$ \mathbb{L}
$x, y, z : \mathbb{R}$ $radius : \mathbb{R}$	$d:=sqrt((s_1.x-s_2.x)^2+(s_1.y-s_2.y)^2+(s_1.z-s_2.z)^2)s_1.radius -s_2.radius \ l:=distance(s_1, s_2) - 2 \cdot s_2.radius \le 0$
\\ Invariáns: ra	adius >= 0.0

A pont egy speciális gömb, aminek a sugara nulla.

Point

Invariáns: radius = 0.0

Az alábbi műveletek mind értelmesek: distance(s_1 , s_2), distance(s_1 , p_2), distance(p_1 , p_2), contains(s_1 , s_2), contains(s_1 , s_2), contains(p_1 , p_2) ahol s_1 , s_2 : Sphere, p_1 , p_2 : Point

Altipusos modell

Sphere

Itt az x, y, z adattagok láthatósága védett (#) kell legyen, hogy a Point-ban ezekre közvetlenül hivatkozhasson.

#x : double

y : double

#z : double

- radius : double

+ Sphere(double,double,double)

+ distance(s:Sphere) : double o

+ contains(s:Sphere) : bool ○

Invariáns: radius >= 0.0

return sqrt($(x-s.x)^2+(y-s.y)^2+(z-s.z)^2$) – radius – s.radius

return distance(s) + 2· s.radius ≤ 0

Ha s:Sphere és p:Point, akkor s = p értékadás értelmes.

Emiatt lehet meghívni a Sphere metódusait az alábbi módokon is: b.distance(p), b.contains(p), ahol b:Sphere.

A q.distance(b) és q.contains(b) (q:Point) hívások viszont az öröklés miatt használhatók, de egyszerre mindkét tulajdonságot kihasználják a q.distance(p), q.contains(p) hívások.

Point

Invariáns: radius = 0.0

- + Point()
- + Point(double,double,double)
- + set(double,double,double) : void

```
Sphere
```

```
# x : double
# y : double
# z : double
- radius : double
```

+ Sphere(double,double,double,double) class Sphere { + distance(s:Sphere) : double public: + contains(s:Sphere) : bool enum Errors { NEGATIVE RADIUS }; Sphere (double x, double y, double z, double r) : x(x), y(y), z(z) { if(r<0) throw NEGATIVE RADIUS; radius = r;</pre> double distance(const Sphere &s)const { **return** sqrt(pow(_x-s._x,2) + pow(_y-s._y,2) + pow(_z-s._z,2)) - radius - s. radius; bool contains(const Sphere &s) const { return distance(s) + 2*s. radius <= 0;</pre> protected: **double** _x, _y, _z;

private:

};

double radius;

Point

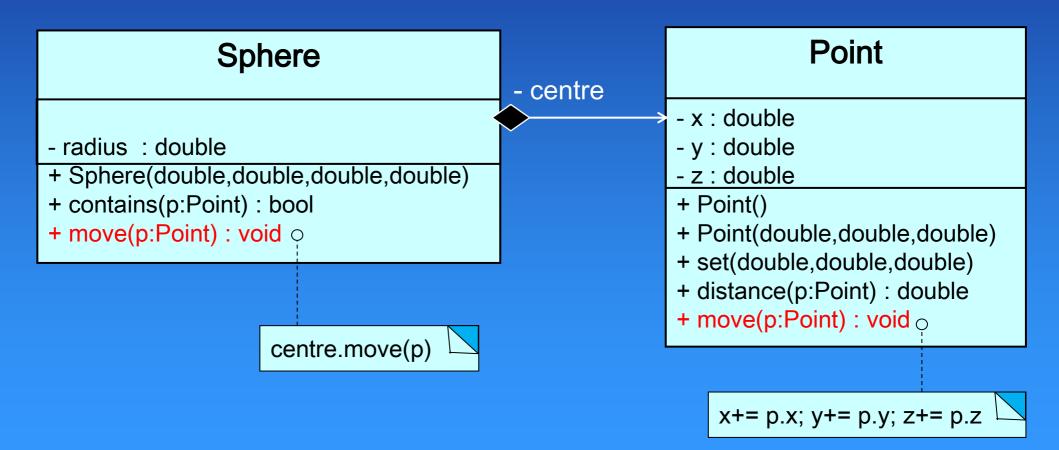
- + Point()
- + Point(double,double,double)
- + set(double,double,double) : void

```
class Point : public Sphere {
  public:
    Point():Sphere(0.0,0.0,0.0,0.0) {}
    Point(double x, double y, double z):Sphere(x,y,z,0.0) {}
    void set(double x, double y, double z) {_x = x; _y = y; _z = z;}
};
```

Ugyanaz a fő program

```
int main()
   double x,y,z,r;
   ... // x,y,z,r beolvasása
   try{
       Sphere s(x,y,z,r);
   catch (Sphere::Errors err) {
       if(err==Sphere::NEGATIVE RADIUS) cout << ...;</pre>
   int n; cin >> n;
   vector<Point> v(n);
   for(int i=0; i<n; ++i) {
       cout << i+1 << ". points:" << endl;
       ... // x,y,z beolvasása
       v[i].set(x,y,z);
   int c = 0;
   for (int i=0; i<n; ++i) {</pre>
       if(s.contains(v[i])) ++c;
   cout << "result: " << c << endl;</pre>
   return 0;
```

Kompozíciós modell eltolás művelettel kiegészítve



Származtatásos modell eltolás művelettel kiegészítve

Point

- x : double
- y : double
- z : double
- + Point()
- + Point(double,double,double)
- + set(double,double,double) : void
- + distance(p:Point) : double
- + move(p:Point) : void O-

x+= p.x; y+= p.y; z+= p.z

Sphere

radius : double

- + Sphere(double,double,double)
- + contains(p:Point) : bool
- + distance(p:Point) : double O-----

return Point::distance(p) ≦ radius

return Point::distance(p) – radius



Altípusos modell eltolás művelettel kiegészítve

Sphere

x : double # y : double # z : double -radius : double

+ Sphere(double,double,double)

+ distance(s:Sphere) : double

+ contains(s:Sphere) : bool

+ move(s:Sphere) : void O--

x+= p.x; y+= p.y; z+= p.z

Point

- + Point()
- + Point(double,double,double)
- + set(double,double,double) : void