

Programozási tételek újrafelhasználható osztálysablon könyvtára

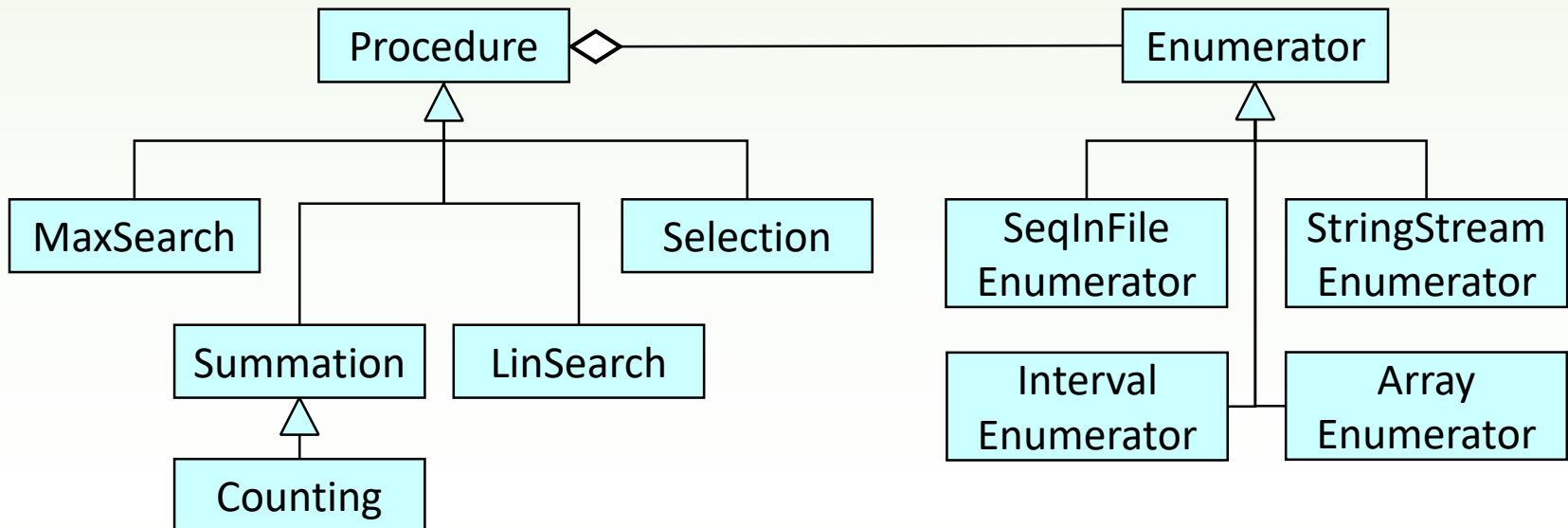
Gregorics Tibor

gt@inf.elte.hu

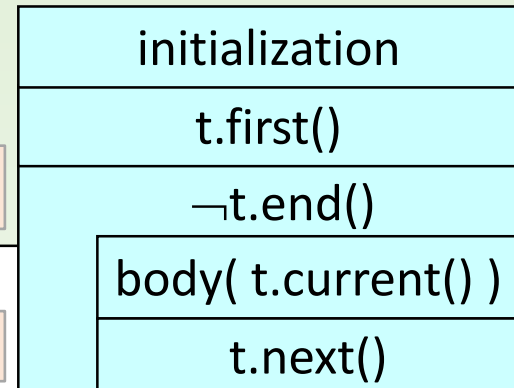
<http://people.inf.elte.hu/gt/oep>

Cél

- ❑ Legyen egy **programozási tételeket általánosan leíró kódkönyvtár** (osztálysablon könyvtár), amely felhasználásával a visszavezetéssel tervezett programjainkat minimális erőfeszítéssel (ciklusok írása nélkül) implementálhatjuk.
- ❑ Egy feladat megoldása egy ún. **tevékenység objektum** lesz, amelynek
 1. osztályát a kódkönyvtár egy osztálysablonjából **származtatjuk**,
 2. megadva annak **sablon paramétereit** és felülírva a **metódusait**,
 3. és futási időben egy **felsoroló objektumot** is csatolunk hozzá.



Programozási tételek ősiklusa



```
template <typename Item>
```

```
void run()
```

```
{
```

```
    if ( _enor == nullptr ) throw MISSING_ENUMERATOR ;
```

```
    init();
```

```
    for ( first(); loopCond(); _enor->next() )
```

```
    {
```

```
        body(_enor->current());
```

```
    }
```

```
}
```

felsorolt elemek típusa

felsoroló objektumra mutató pointer

default: !_enor->end() && whileCond(_enor->current())

default: true

default: _enor->first()

Ez a run() függvény bármelyik programozási tétel algoritmusának leírására alkalmas, ha az init(), és a body() függvényeket megfelelő módon definiáljuk. Lehetőség van a first(), a whileCond(), vagy akár a teljes loopCond() függvény felüldefiniálására is, ha a felsorolás indításán, illetve leállításán módosítani akarunk. Az Item sablonparaméterrel a felsorolt elemek típusát adjuk meg.

Programozási tételek ősosztálya

template <typename Item>

class Procedure {

protected:

Enumerator<Item> *_enor;

Procedure():_enor(**nullptr**){}

virtual void init()= 0;

virtual void body(**const** Item& current) = 0;

virtual void first() {_enor->first();}

virtual bool whileCond(**const** Item& current) **const** { **return true**;}

virtual bool loopCond() **const**

{ **return** !_enor->end()&& whileCond(_enor->current());}

public:

enum Exceptions { MISSING_ENUMERATOR };

virtual void run() **final**;

virtual void addEnumerator(Enumerator<Item>* en) **final** { _enor = en;}

virtual ~Procedure(){}

};

felsorolt elemek típusa

felsoroló objektumra mutató pointer

a run() felüldefiniálható metódusai

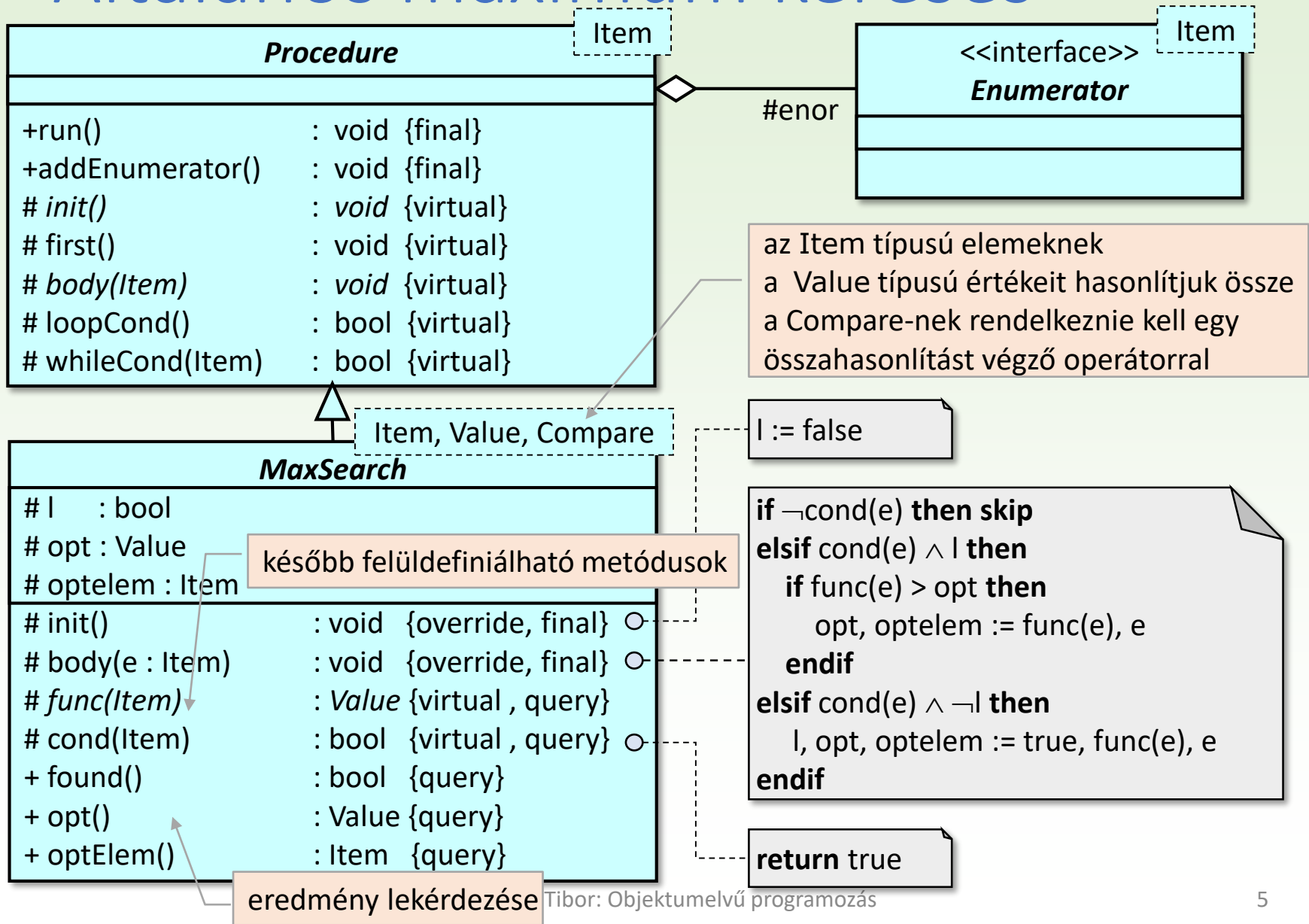
nem írható felül a leszármazott osztályokban

konkrét felsoroló objektum hozzáadása

procedure.hpp

az ős ciklust tartalmazó run(), amelyik felüldefiniálható metódusokat hív meg (ld sablon-függvény tervminta)

Általános maximum keresés



Maximum keresés osztálya

```
template <typename Item, typename Value = Item,  
          typename Compare = Greater<Value> >
```

```
class MaxSearch : public Procedure<Item>  
{
```

```
    protected:
```

```
        bool    _l;  
        Item    _optelem;  
        Value    _opt;  
        Compare _better;
```

a maximum vagy minimum kereséshez biztosítja a megfelelő összehasonlítást

két Value típusú értéket összehasonlító adattag

```
    void init() override final { _l = false; }  
    void body(const Item& e) override final;  
    virtual Value func(const Item& e) const = 0;  
    virtual bool cond(const Item& e) const { return true; }
```

```
    public:
```

```
        bool found;  
        Value opt();  
        Item optElem;
```

```
};
```

```
template <typename Item, typename Value, typename Compare>  
void MaxSearch<Item, Value, Compare>::body(const Item& e)
```

```
{  
    if ( !cond(e) ) return;  
    Value val = func(e);  
    if (_l){  
        if (_better(val, _opt)){ ... }  
    }  
    else { ... }  
}
```

A Compare helyébe olyan típusnak kell kerülnie, amely úgy implementálja az operator(,)-t, hogy a _better objektum a kívánt módon hasonlítsa össze a val és az _opt értékeket.

maxsearch.hpp

Összehasonlító osztályok

```
template <typename Value>
```

```
class Greater{
```

```
public:
```

```
    bool operator()(const Value& l, const Value& r)
```

```
    {
```

```
        return l > r;
```

```
    }
```

```
};
```

Ha a `_better` típusa a `Greater<int>`,
akkor `_better(2,5)` azonos a `2>5` értékével.

```
template <typename Value>
```

```
class Less{
```

```
public:
```

```
    bool operator()(const Value& l, const Value& r)
```

```
    {
```

```
        return l < r;
```

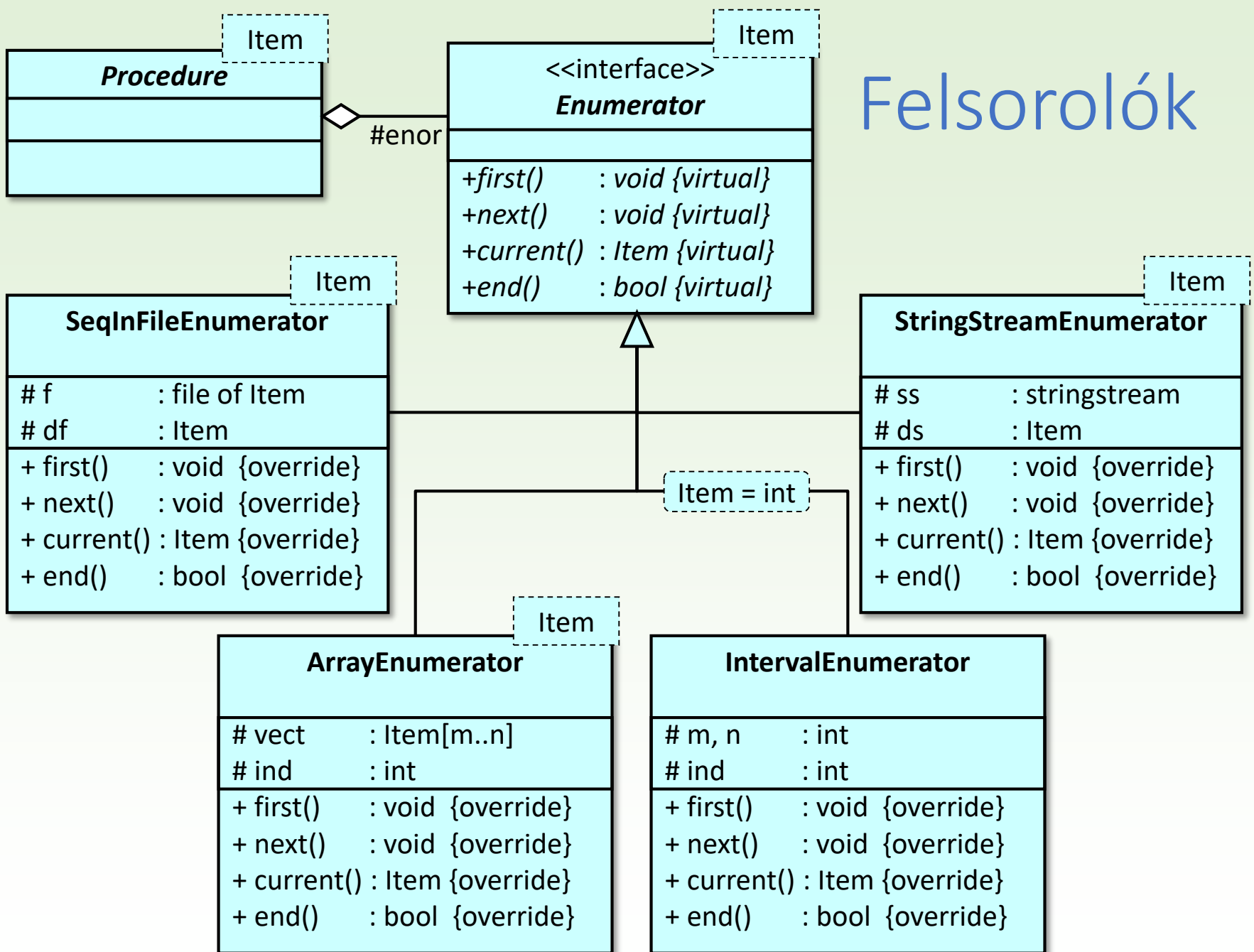
```
    }
```

```
};
```

Ha a `_better` típusa a `Less<int>`,
akkor `_better(2,5)` azonos a `2<5` értékével.

maxsearch.hpp

Felsorolók



Intervallum és Tömb felsorolója

```
class IntervalEnumerator : public Enumerator<int> {  
    protected:  
        int    _m, _n;  
        int    _ind;  
    public:  
        IntervalEnumerator(int m, int n):_m(m), _n(n){}  
        void first()           override { _ind = m; }  
        void next()           override { ++_ind; }  
        bool end()           const override { return _ind>_n; }  
        Item current() const override { return _ind; }  
};
```

intervalenumerator.hpp

```
template <typename Item>  
class ArrayEnumerator : public Enumerator<Item> {  
    protected:  
        const std::vector<Item> *_vect;  
        unsigned int _ind;  
    public:  
        ArrayEnumerator(const std::vector<Item> *v):_vect(v){}  
        void first()           override { _ind = 0; }  
        void next()           override { ++_ind; }  
        bool end()           const override { return _ind>=_vect->size(); }  
        Item current() const override { return (*_vect)[_ind]; }  
};
```

arrayenumerator.hpp

Szekvenciális inputfájl felsorolója

```
template <typename Item>
class SeqInFileEnumerator : public Enumerator<Item>{
protected:
    std::ifstream _f;
    Item          _df;
public:
    enum Exceptions { OPEN_ERROR };

    SeqInFileEnumerator(const std::string& str){
        _f.open(str);
        if(_f.fail()) throw OPEN_ERROR;
    }
    void first()          override { _f >> _df; }
    void next()           override { _f >> _df; }
    bool end()            const override { return _f.fail(); }
    Item current() const override { return _df; }
};
```

Az Item-re legyen értelmes az istream& **operator**>>

seqinfileenumerator.hpp

A kód-könyvtárban ez az osztály valójában ennél összetettebb:

- egyrészt rendelkezik egy olyan (Item = char) specializációval, amely kikapcsolja az elválasztó jeleket (white-space) figyelmen kívül hagyó mechanizmust,
- másrészt soronkénti olvasás esetén az üres sorokat figyelmen kívül hagyja.

StringStreamEnumerator

```
template <typename Item>
class StringStreamEnumerator : public Enumerator<Item> {
protected:
    std::stringstream _ss;
    Item               _df;
public:
    StringStreamEnumerator(std::stringstream& ss) {_ss << ss.rdbuf(); }

    void first()           override { _ss >> _df; }
    void next()            override { _ss >> _df; }
    bool end()             const override { return !_ss; }
    Item current() const override { return _df; }
};
```

Az Item-re legyen értelmes
az istream& **operator**>>

stringstreamenumerator.hpp

1. Feladat

Adott egy egész számokat tartalmazó **szöveges állomány**. Keressük meg ebben a **legnagyobb páratlan számot**!

$A : f:\text{infile}(\mathbb{Z}) , l:\mathbb{L}, \text{max}:\mathbb{Z}$

$Ef : f = f_0$

$Uf : l, \text{max} = \mathbf{MAX}_{e \in f_0} e$
e páratlan

Feltételes maximum keresés

$t:\text{enor}(E) \sim f:\text{infile}(\mathbb{Z})$ felsorolása

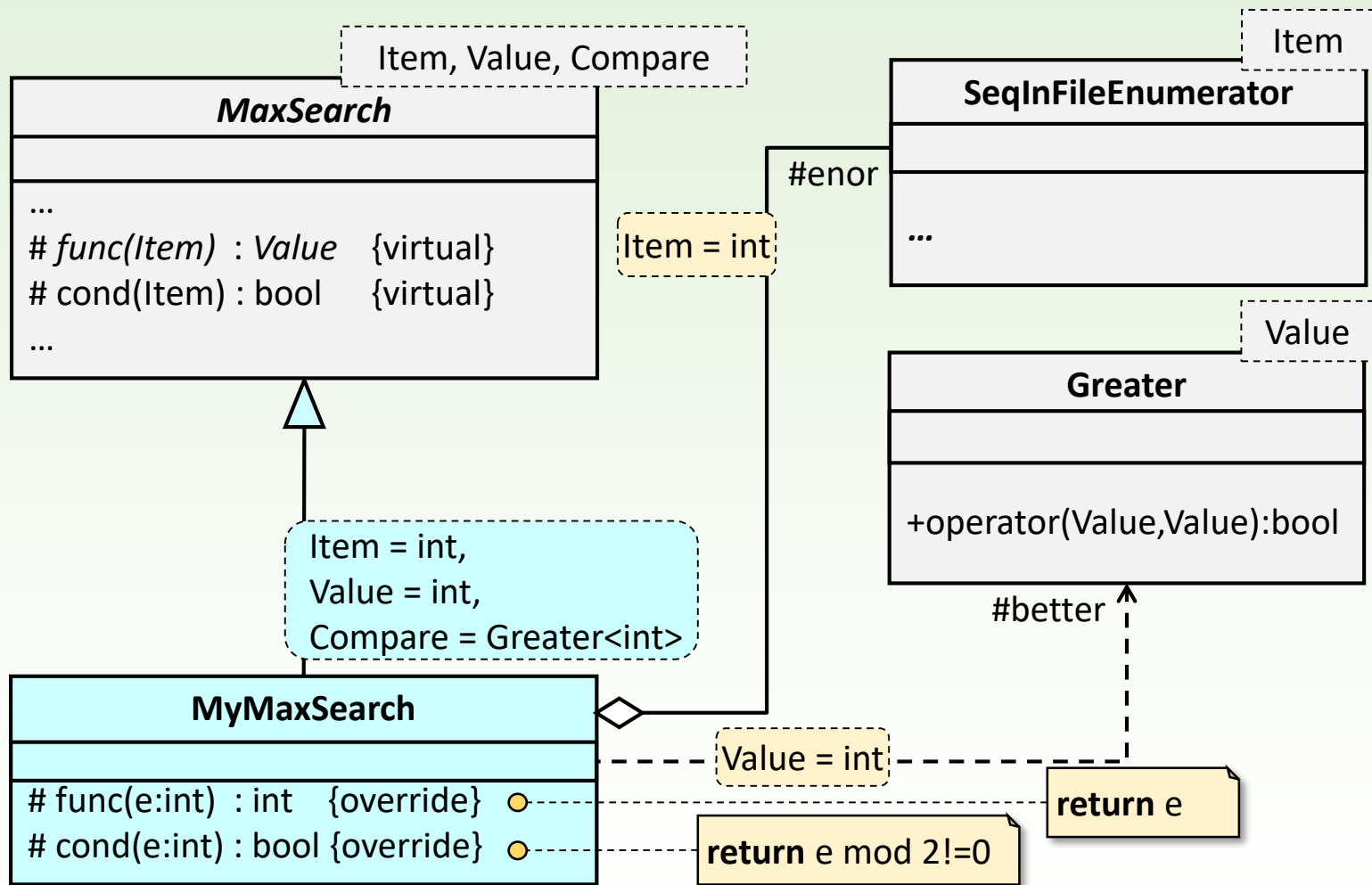
$H, > \sim \mathbb{Z}, >$

$f(e) \sim e$

$\text{felt}(e) \sim e \text{ páratlan}$

E	/ Item	~	\mathbb{Z}	/ int
H	/ Value	~	\mathbb{Z}	/ int
f(e)	/ func(e)	~	e	
felt(e)	/ cond(e)	~	e páratlan	/ e%2!=0

Megoldás osztálydiagramja



A megvalósításhoz készített kód

```
class MyMaxSearch : public MaxSearch<int>{  
    protected:  
        int func(const int& e) const override { return e;}  
        bool cond(const int& e) const override { return e%2!=0;}  
};
```

12 -5 23 input.txt
44 130 56 3
-120

```
int main(){  
    try {  
        MyMaxSearch pr;  
        SeqInFileEnumerator<int> enor("input.txt");  
        pr.addEnumerator(&enor);  
        pr.run();  
  
        if (pr.found())  
            cout << "The greatest odd integer:" << pr.optElem();  
        else  
            cout << "There is no odd integer!";  
    } catch (SeqInFileEnumerator<int>::Exceptions ex){  
        if (SeqInFileEnumerator<int>::OPEN_ERROR == ex )  
            cout << "Wrong file name!";  
    }  
    return 0;  
}
```

tevékenység objektum

felsoroló objektum

működés

2. Feladat

Egy forgalomszámlálás során egy hónapon keresztül számolták, hogy óránként **hány utas lép be egy adott metróállomás területére**. (A méréseket hétfőn kezdték, de nem minden nap és nem minden órában végeztek megfigyelést.)

Az időt négyjegyű számmal kódolták, amelynek első két jegye a mérés napját mutatja (pontosabban azt, hogy hányadik napja ez a megfigyelésnek), utolsó két jegye a nap azon óráját mutatja, amikor a mérés született. A méréseket egy **szöveges állományban rögzítették idő kód-létszám párok formájában**.

A hétvégi napok közül (melyik nap melyik órájában) **mikor léptek be az állomás területére legkevesebben?**

```
0108 23    input.txt
0112 44
0116 130
0207 120
...
```

Megoldás specifikációja

$A : f:\text{infile}(\text{Mérés}) , l:\mathbb{L}, \text{min}:\mathbb{N}, \text{elem}: \text{Mérés}$
 $\text{Mérés} = \text{rec}(\text{időpont} : \mathbb{N}, \text{létszám} : \mathbb{N})$

$Ef : f = f_0$

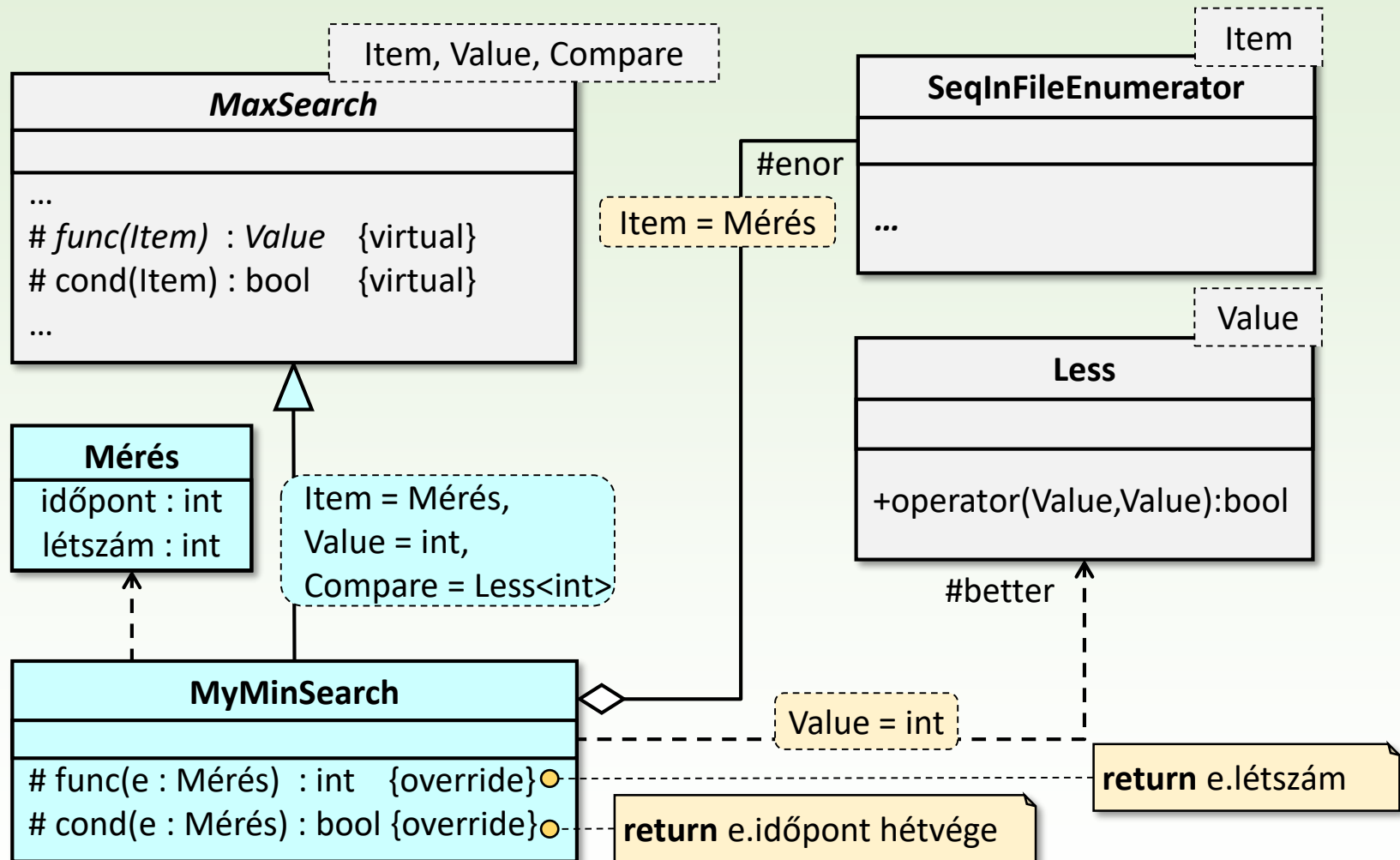
$Uf : l, \text{min}, \text{elem} = \mathbf{MIN}_{e \in f_0} \text{e.létszám}$
 e.időpont hétvége

Feltételes maximum keresés

$t:\text{enor}(\text{Item})$	\sim	$f:\text{infile}(\text{Mérés})$ felsorolása
Item	\sim	Mérés
$\text{Value}, >$	\sim	$\mathbb{N}, <$
$\text{func}(e)$	\sim	$e.\text{létszám}$
$\text{cond}(e)$	\sim	$e.\text{időpont hétvége}$

$e.\text{időpont}/100\%7 == 6 \mid\mid e.\text{időpont}/100\%7 == 0$

Megoldás osztálydiagramja



Elemi típus (Mérés)

```
0108 23      input.txt
0112 44
0116 130
0207 120
...
```

```
struct Measurement{
    int timestamp;
    int number;

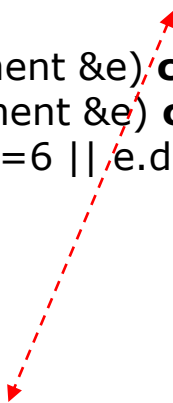
    int day() const { return timestamp/100; }
    int hour() const { return timestamp%100; }
};
```

a szekvenciális inputfájl Pair típusú
elemeinek felsorolásához kell

```
istream& operator>>(istream& f, Measurement& df)
{
    f >> df.timestamp >> df.number;
    return f;
}
```

Főprogram

```
class MyMinSearch: public MaxSearch< Measurement, int, Less<int> > {  
    protected:  
        int func(const Measurement &e) const override { return e.number; }  
        bool cond(const Measurement &e) const override  
            { return e.day()%7==6 || e.day()%7==0; }  
};  
int main()  
{  
    try {  
        MyMinSearch pr;  
        SeqInFileEnumerator< Measurement > enor("input.txt");  
        pr.addEnumerator(&enor);  
        pr.run();  
  
        if (pr.found()){  
            Pair p = pr.optElem();  
            cout << "The least busy hour was the " << p.hour()  
                << ". hour of the " << p.day() << ". day when "  
                << pr.opt() << " people stepped into the station.\n";  
        } else cout << "There is no weekend data.\n";  
    } catch(Exceptions ex){  
        if (OPEN_ERROR == ex ) cout << "File open error!";  
    }  
    return 0;  
}
```



3. Feladat

Egy szöveges állomány sorai recepteket tartalmaznak, ahol egy recept az étel nevéből (sztring) és a hozzávalók felsorolásából áll. Egy hozzávalót anyagnévvel (sztring), mennyiséggel (szám), és mértékegységgel (sztring) adunk meg.

Példa egy sorra:

tejbegríz tej 1 liter búzadara 13 evőkanál vaj 6 dkg cukor 5 evőkanál

Hány receptnek hozzávalója a cukor?

$A : f:\text{infile}(\text{Recept}) , db:\mathbb{N}$

$\text{Recept} = \text{rec}(\text{név} : \text{String}, \text{hozzávalók} : \text{Hozzávaló}^*)$

$\text{Hozzávaló} = \text{rec}(\text{anyag} : \text{String}, \text{mennyiség} : \mathbb{N}, \text{mérték} : \text{String})$

$Ef : f = f_0$

$Uf : db = \sum_{e \in f_0} \mathbf{1}_{\text{van_cukor}(e)}$

részfeladat: van-e a hozzávalók között cukor?

$\text{van_cukor}(e) = \text{SEARCH}_{i=1..|e.\text{hozzávalók}|} e.\text{hozzávalók}[i].\text{anyag}=\text{cukor}$

egy sor elemei alapján

Számlálás

$t:\text{enor}(\text{Item}) \sim f:\text{infile}(\text{Recept})$
felsorolása

$\text{Item} \sim \text{Recept}$

$\text{cond}(e) \sim \text{van_cukor}(e)$

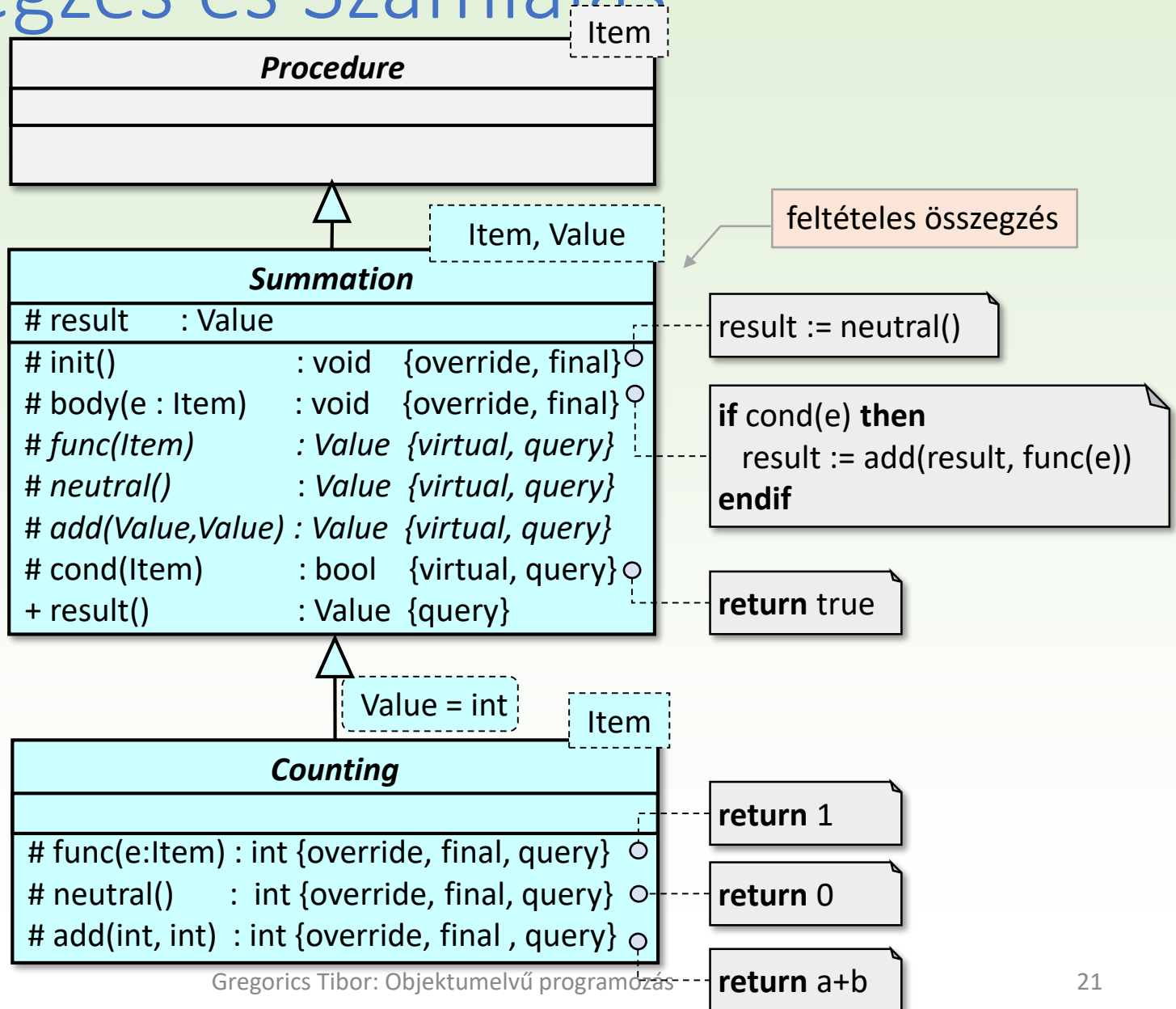
Lineáris keresés

$t:\text{enor}(\text{Item}) \sim e.\text{hozzávalók} : \text{Hozzávaló}^*$
felsorolása

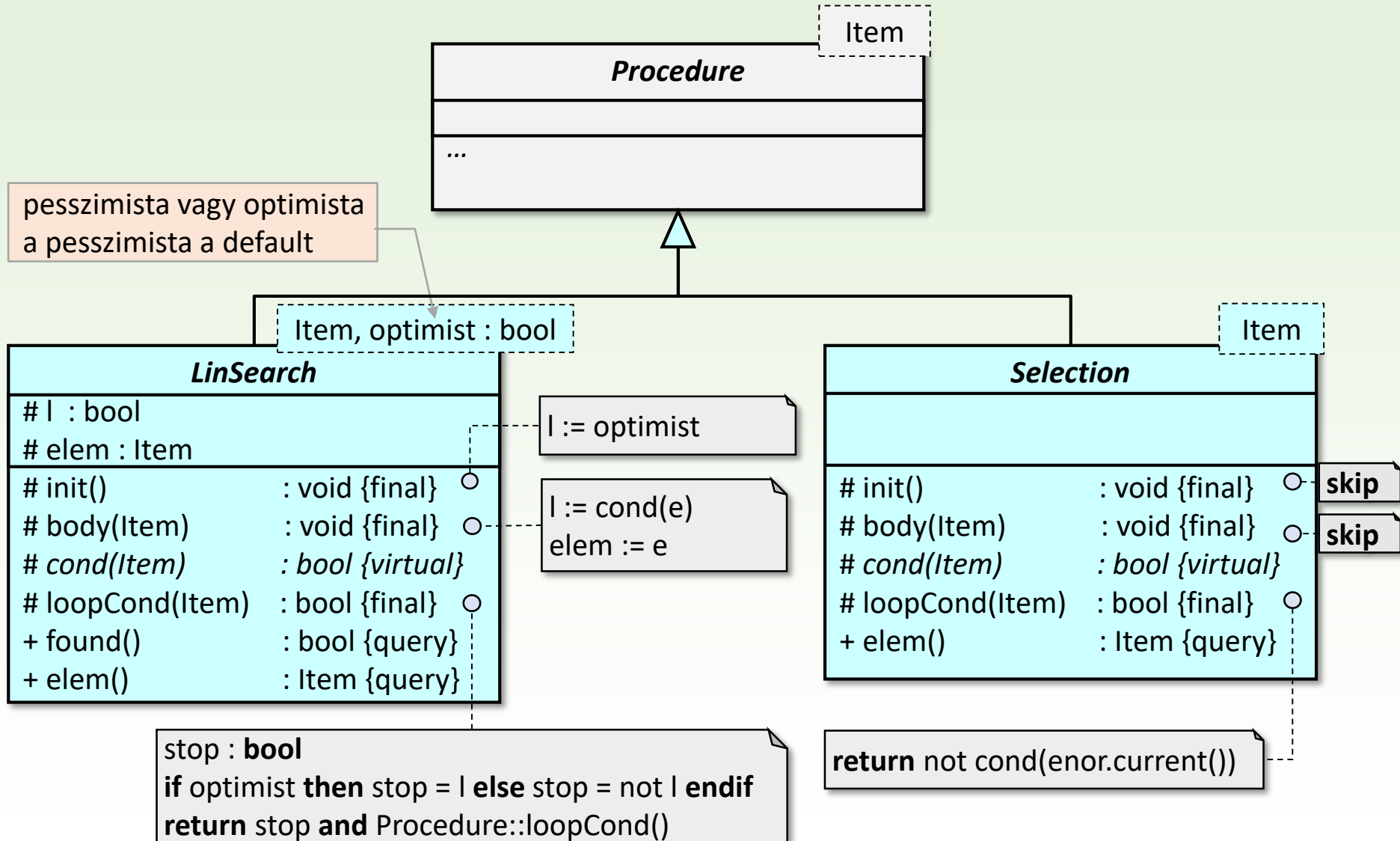
$\text{Item} \sim \text{Hozzávaló}$

$\text{cond}(e) \sim e.\text{hozzávalók}[i].\text{anyag}=\text{cukor}$

Összegzés és Számlálás



Lineáris keresés és kiválasztás



1. megoldás kezdete

main

```
pr : MyCounting
enor : SeqInFileEnumerator<Recipe> ("input.txt")
pr.addEnumerator(&enor)
pr.run()
return pr.result()
```

Item = Recipe

MyCounting : Counting

```
# cond(e:Recipe) : bool {override}
```

```
pr : MyLinSearch
enor : ArrayEnumerator<Ingredient> (e.vect)
pr.addEnumerator(&enor)
pr.run()
return pr.found()
```

Item = Ingredient

MyLinSearch : LinSearch

```
# cond(e: Ingredient):bool {override}
```

```
return e.substance="sugar"
```

Recipe

```
name : string
vect : Ingredient[*]
```

operator>>(is:istream, e:Recipe)

```
line : string
getline(is, line)
ss : stringstream(line)
ss >> e.name
„át kell másolni a hozzávalókat
az ss-ből az e.vect tömbbe”
```

Ingredient

```
substance : string
quantity : int
unit : string
```

operator>>(is:istream, e:Ingredient)

```
is >> e.substance >> e.quantity >> e.unit
```

Sorozatok (vector, ostream) előállítása összegzéssel

A **másolások**, **listázások**, **kiválogatások** összegzésként történő előállítása érdekében készült el a Summation sablon két specializációja:

- ❑ Az eredmény típusát leíró **Value** kétféleképpen jelölhet sorozat-típust
 - **vector<Value>**, és az eredmény-vector-t vagy a konstruktorral adjuk át az összegzésnek, vagy helyben definiáljuk.
 - **ostream**, és az eredmény-adatfolyam hivatkozását a konstruktorral adjuk át az összegzésnek.
- ❑ A **func()** metódus állítja elő azt az elemet, amelyet az eredmény-sorozathoz kell hozzáfűzni.
 - **Value** \leftarrow **vector<Value>** esetén **Value** func(Item) const
 - **Value** \leftarrow **ostream** esetén **string** func(Item) const
- ❑ A **cond()** metódus felüldefiniálásával lehet elérni, hogy a felsorolt elemeknek csak egy részét vegyük figyelembe (pl. kiválogatásnál).
- ❑ Nem kell a **neutral()** és **add()** műveleteket felüldefiniálni.

Summation specializációk

```
class Summation<Item, vector<Value> > : public Procedure<Item, vector<Value> > {  
  private:  
    std::vector<Value> _result;  
  public:  
    Summation() {}  
    Summation(const std::vector<Value> &v) : _result(v) {}  
  protected:  
    ...  
    virtual Value func(const Item& e) const = 0;  
    virtual bool cond(const Item& e) const { return true; }  
};
```

felüldefiniálható metódusok

```
class Summation<Item, ostream> : public Procedure<Item, ostream> {  
  private:  
    std::ostream *_result;  
  public:  
    Summation(std::ostream *o) : _result(o) {}  
  protected:  
    ...  
    virtual string func(const Item& e) const = 0;  
    virtual bool cond(const Item& e) const { return true; }  
};
```

felüldefiniálható metódusok

Kitérő

Szöveges állományból beolvasott egész számokat hozzáfűzünk egy tömbhöz, majd a tömb páros elemeit kilistázzuk

```
class Concat : public Summation<int, vector<int> >{  
public:  
    Concat(const vector<int> &v) : Summation<int, vector<int> >(v) {}  
protected:  
    int func(const int &e) const override { return e; }  
};  
  
class Write : public Summation<int, ostream > {  
public:  
    Write(ostream* o) : Summation<int, ostream>(o) {}  
protected:  
    string func(const int &e) const override {  
        ostringstream os;  
        os << e << " ";  
        return os.str();  
    }  
    bool cond(const int &e) const override {  
        return e%2==0;  
    }  
};
```

```
vector<int> v = { -17, 42 };  
Concat pr1(v);  
SeqInFileEnumerator<int> enor1("input.txt");  
pr1.addEnumerator(&enor1);  
pr1.run();  
  
Write pr2(&cout);  
ArrayEnumerator<int> enor2(&pr1.result());  
pr2.addEnumerator(&enor2);  
pr2.run();
```

1. megoldás befejezése

main

```
pr : MyCounting
enor : SeqInFileEnumerator<Recipe> ("input.txt")
pr.addEnumerator(&enor)
pr.run()
return pr.result()
```

Item = Recipe

MyCounting : Counting

```
# cond(e:Recipe) : bool {override}
```

```
pr : MyLinSearch
enor : ArrayEnumerator<Ingredient> (e.vect)
pr.addEnumerator(&enor)
pr.run()
return pr.found()
```

Item = Ingredient

MyLinSearch : LinSearch

```
# cond(e:Ingredient):bool {override}
```

```
return e.substance="sugar"
```

operator>>(is:istream, e:Recipe)

```
line : string
getline(is, line)
ss : stringstream(line)
ss >> e.name
pr : Copy
enor: StringStreamEnumerator<Ingredient>(ss)
pr.addEnumerator(&enor)
pr.run()
e.vect := pr.result()
```

Át kell másolni a hozzávalókat az ss-ből az e.vect tömbbe.

Item = Ingredient,
Value = vector<Ingredient>

Copy : Summation

```
# func(e:Ingredient) : Ingredient {override}
```

return e

operator>>(is:istream, e:Ingredient)

```
is >> e.substance >> e.quantity >> e.unit
```

Recipe

```
name : string
vect : Ingredient[*]
```

Ingredient

```
substance : string
quantity : int
unit : string
```

2. megoldás

main

```
pr : MyCounting ○  
enor : SeqInFileEnumerator<Recipe> ("in  
pr.addEnumerator(&enor)  
pr.run()  
return pr.result()
```

Item = Recipe

MyCounting : Counting

```
# cond(e:Recipe) : bool {override} ○
```

return e.has_sugar

operator>>(is:istream, e:Recipe)

```
line : string  
getline(is, line)  
ss : stringstream(line)  
ss >> e.name  
pr : MyLinSearch ○  
enor:StringStreamEnumerator<Ingredient>(ss)  
pr.addEnumerator(&enor)  
pr.run()  
e.has_sugar := pr.found()
```

Nem kell átmásolni a hozzávalókat az ss-ből egy tömbbe ahhoz, hogy megkeressük, van-e cukor köztük.

Ugyanaz a lineáris keresés, mint az 1. megoldásban, csak máshol jelenik meg.

Item = Ingredient

MyLinSearch : LinSearch

```
# cond(e:Ingredient):bool {override} ○
```

return e.substance="sugar"

Recipe ○

```
name : string  
has_sugar : bool
```

Ingredient ○

```
substance : string  
quantity : int  
unit : string
```

operator>>(is:istream, e:Ingredient)

is >> e.substance >> e.quantity >> e.unit

4. Feladat

Egy szöveges állományban aszteroidákról vett megfigyeléseket tárolnak (egy aszteroidáról akár többet is). Minden sor egy megfigyelést tartalmaz: az aszteroida kódját (sztring), egy dátumot (sztring), az aszteroida tömegét (ezer tonnában), az aszteroida távolságát a Földtől (százezer kilométerben).

AXS0076 2015.06.13. 2000 5230

Az állomány aszteroidák kódja szerint növekedően rendezett.

Listázzuk ki azokat az aszteroidákat a legnagyobb mért tömegükkel, amelyek minden megfigyeléskor 1 milliárd kilométernél közelebb voltak a Földhöz!

$A : f:\text{infile}(\text{Megfigyelés}) , \text{cout}:\text{outfile}(\text{String} \times \mathbb{N})$

$\text{Megfigyelés} = \text{rec}(\text{azon} : \text{String}, \text{dátum} : \text{String}, \text{tömeg} : \mathbb{N}, \text{távolság} : \mathbb{N})$

$A : t:\text{enor}(\text{Aszteroida}) , \text{cout}:\text{outfile}(\text{String} \times \mathbb{N})$

$\text{Aszteroida} = \text{rec}(\text{azon} : \text{String}, \text{tömeg} : \mathbb{N}, \text{közel} : \mathbb{L})$

$Ef : t = t_0$

$Uf : \text{cout} = \bigoplus_{\substack{e \in t_0 \\ e.\text{közel}}} \langle (e.\text{azon}, e.\text{tömeg}) \rangle$

Összegzés (kiválogatás)

$t:\text{enor}(\text{Item}) \sim t:\text{enor}(\text{Aszteroida})$

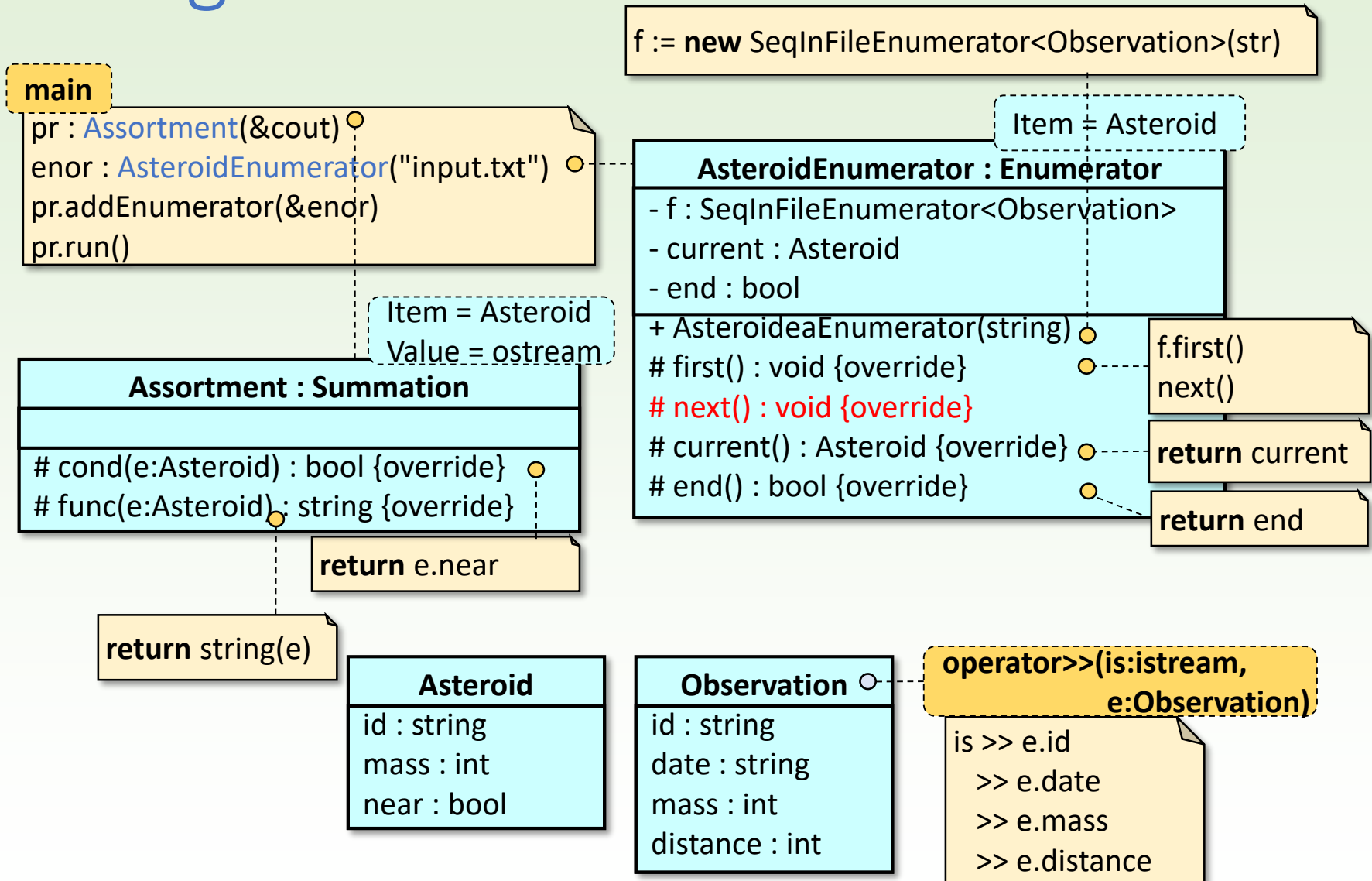
$\text{Item} \sim \text{Aszteroida}$

$\text{Value}, +, 0 \sim (\text{String} \times \mathbb{N})^*, \bigoplus, \langle \rangle$

$\text{func}(e) \sim (e.\text{azon}, e.\text{tömeg})$

$\text{cond}(e) \sim e.\text{közel}$

Megoldás terve



Egy aszteroida adatainak olvasása

A next() metódus kiszámolja egy aszteroida legnagyobb tömegét, valamint azt, hogy mindig közelebb volt-e a Földhöz, mint 1 milliárd kilométer.

$A : f:\text{inFile}(\text{Megfigyelés}), e:\text{Megfigyelés}, st:\text{Status}, akt:\text{Aszteroida}, vége:\mathbb{L}$

$\text{Megfigyelés} = \text{rec}(\text{azon}:\text{String}, \text{dátum}:\text{String}, \text{tömeg}:\mathbb{N}, \text{távolság}:\mathbb{N})$

$\text{Aszteroida} = \text{rec}(\text{azon}:\text{String}, \text{tömeg}:\mathbb{N}, \text{közel}:\mathbb{L})$

$Ef : f = f' \wedge e = e' \wedge st = st'$

$Uf : vége = (st' = \text{abnorm}) \wedge (\neg vége \rightarrow akt.\text{azon} = e'.\text{azon} \wedge$

$akt.\text{tömeg}, st, e, f = \text{MAX}_{e \in (e', f')}^{e.\text{azon} = akt.\text{azon}} e.\text{tömeg} \wedge$

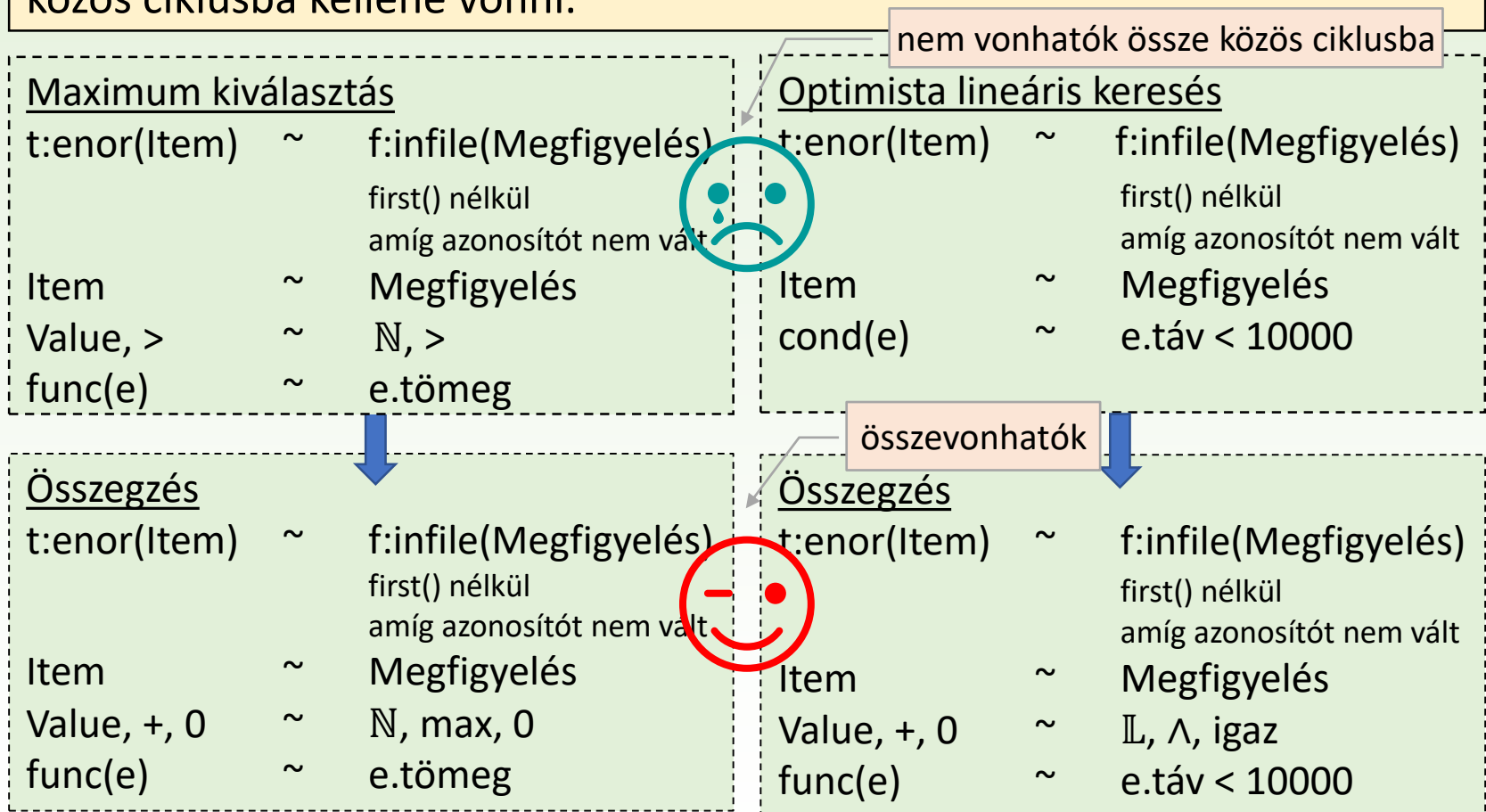
$akt.\text{közel}, st'', e'', f'' = \text{VSEARCH}_{e \in (e', f')}^{e.\text{azon} = akt.\text{azon}} (e.\text{távolság} < 10000)$

ennek a két felsorolásnak ugyanazon állapotból kell indulni: nem végezhetők el szekvenciában

a két feldolgozás eltérő állapotokban állhat le: ezt is szinkronizálni kell

Két programozási tétel összevonása

Egy aszteroida legnagyobb tömegét megadó maximum kiválasztást, és a Földhöz való közelségét kiszámoló optimista lineáris keresést közös ciklusba kellene vonni.



next() metódus terve

next

```
end := f.end()  
if end then return endif  
current.id := f.current().id  
pr : DoubleSummation (current.id)  
pr.addEnumerator(f)  
pr.run()  
current.mass := pr.result().mass  
current.near := pr.result().near
```

Asteroid

id : string
mass : int
near : bool

Observation

id : string
date : string
mass : int
distance : int

Item = Observation
Value = Result

Result

mass : int
near : bool

Result(int, bool)

DoubleSummation : Summation

- id : string

+ DoubleSummation(str : string)

func(**const** Observation& e) : Result {override}

neutral() : Result {override}

add(Result a, Result b) : Result {override}

first() : void {override}

whileCond(Observation e) : bool {override}

id = str

return Result(
e.mass,
e.distance < 10000)

return Result(0, true)

return Result(
max(a.mass, b.mass),
a.near **and** b.near)

skip

return e.id = id