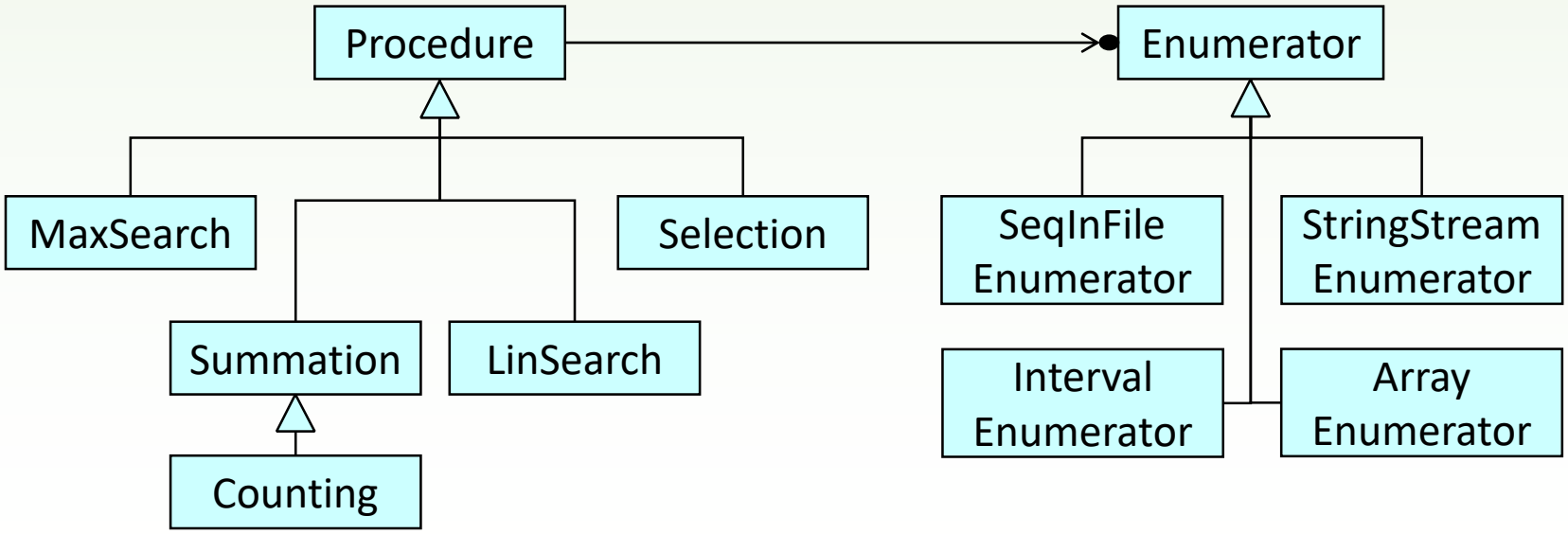
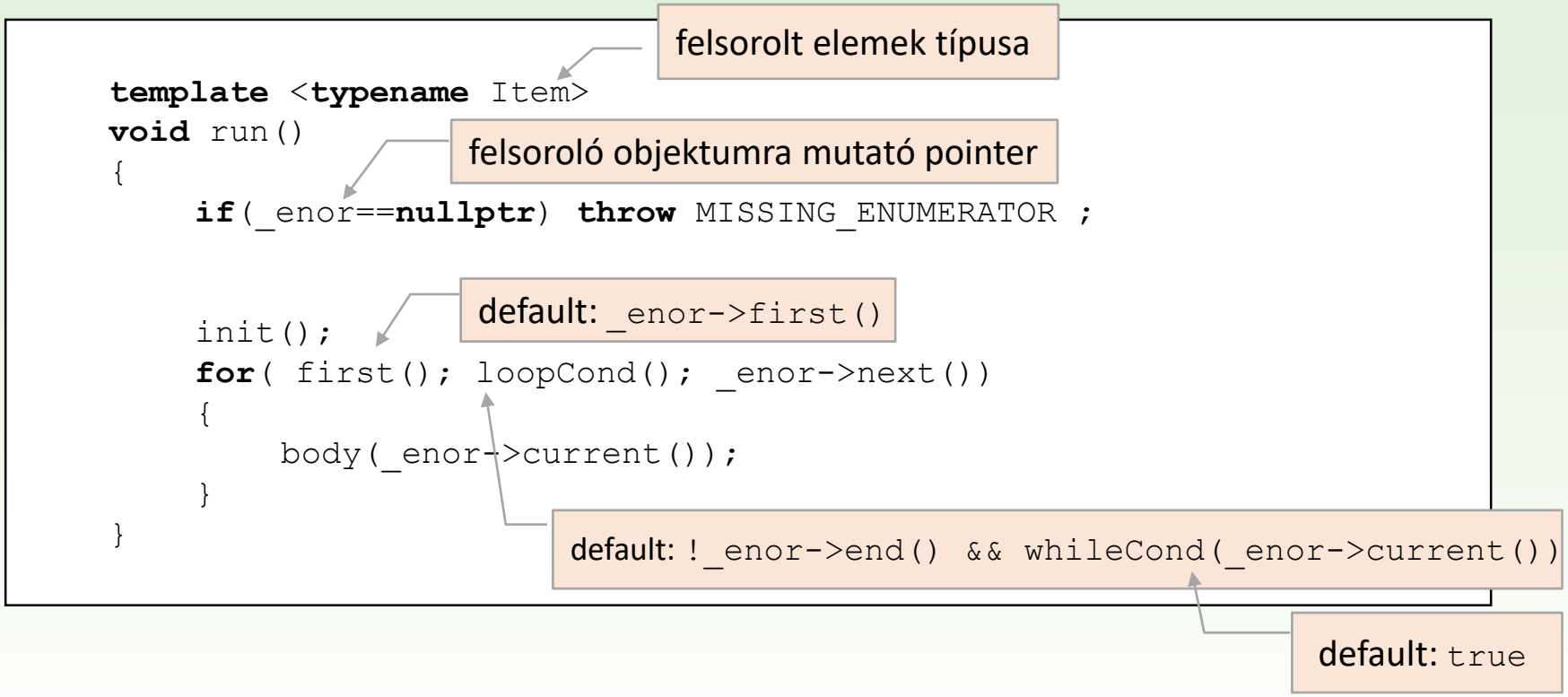
Programozási tételek újrafelhasználható osztálysablon könyvtára

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 legyen,
  1. amelynek osztályát származtatjuk a kódkönyvtár egy osztályából,
  2. amely sablon paramétereit megadjuk és metódusait felülírjuk,
  3. majd futási időben egy felsoroló objektumot is csatolunk hozzá.



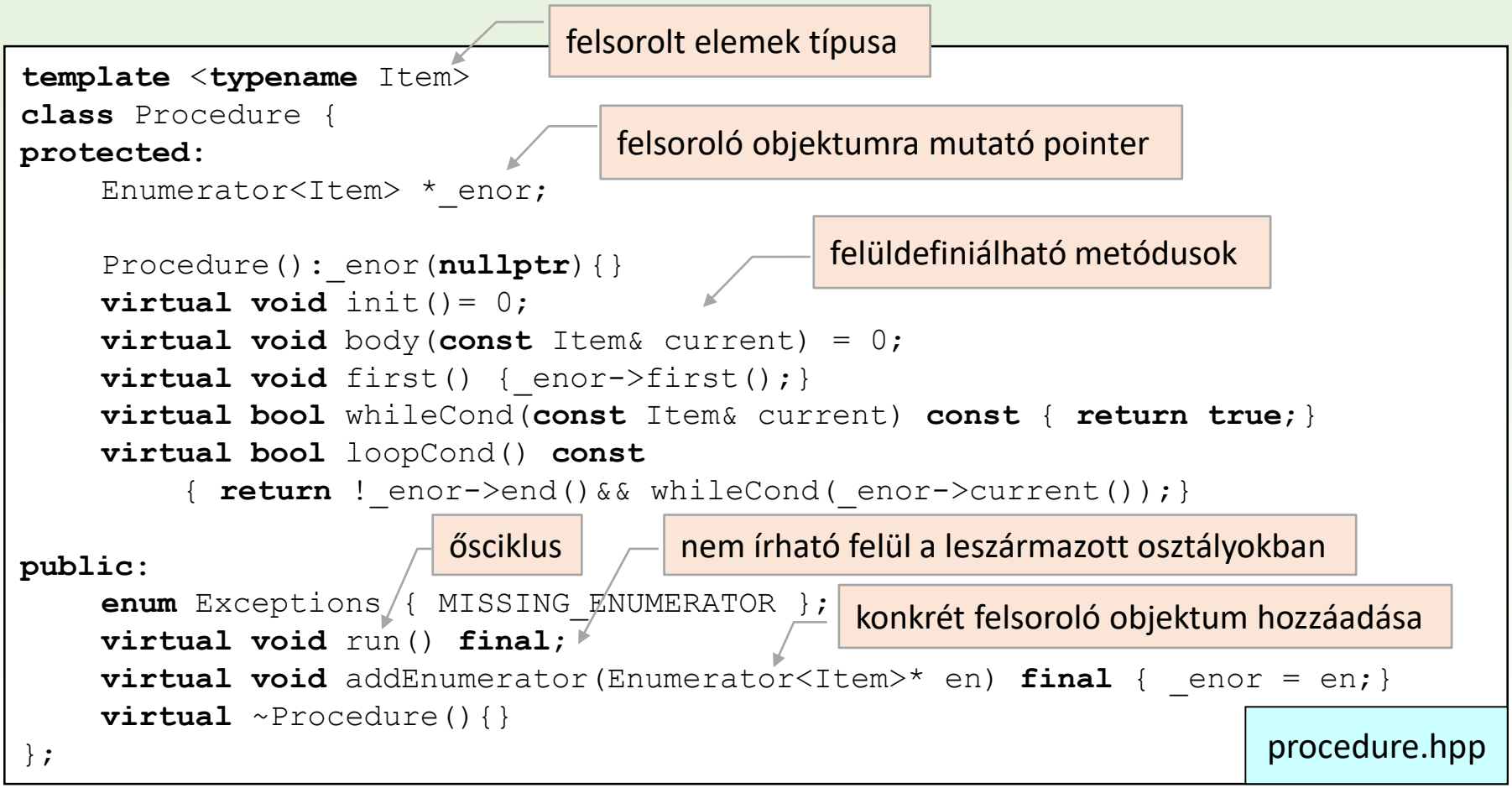
Programozási tételek ősciklusa




```
template <typename Item>
void run()
{
     if(_enor==nullptr) throw MISSING_ENUMERATOR ;

     init();
     for( first(); loopCond(); _enor->next())
     {
          body(_enor->current());
     }
}
```

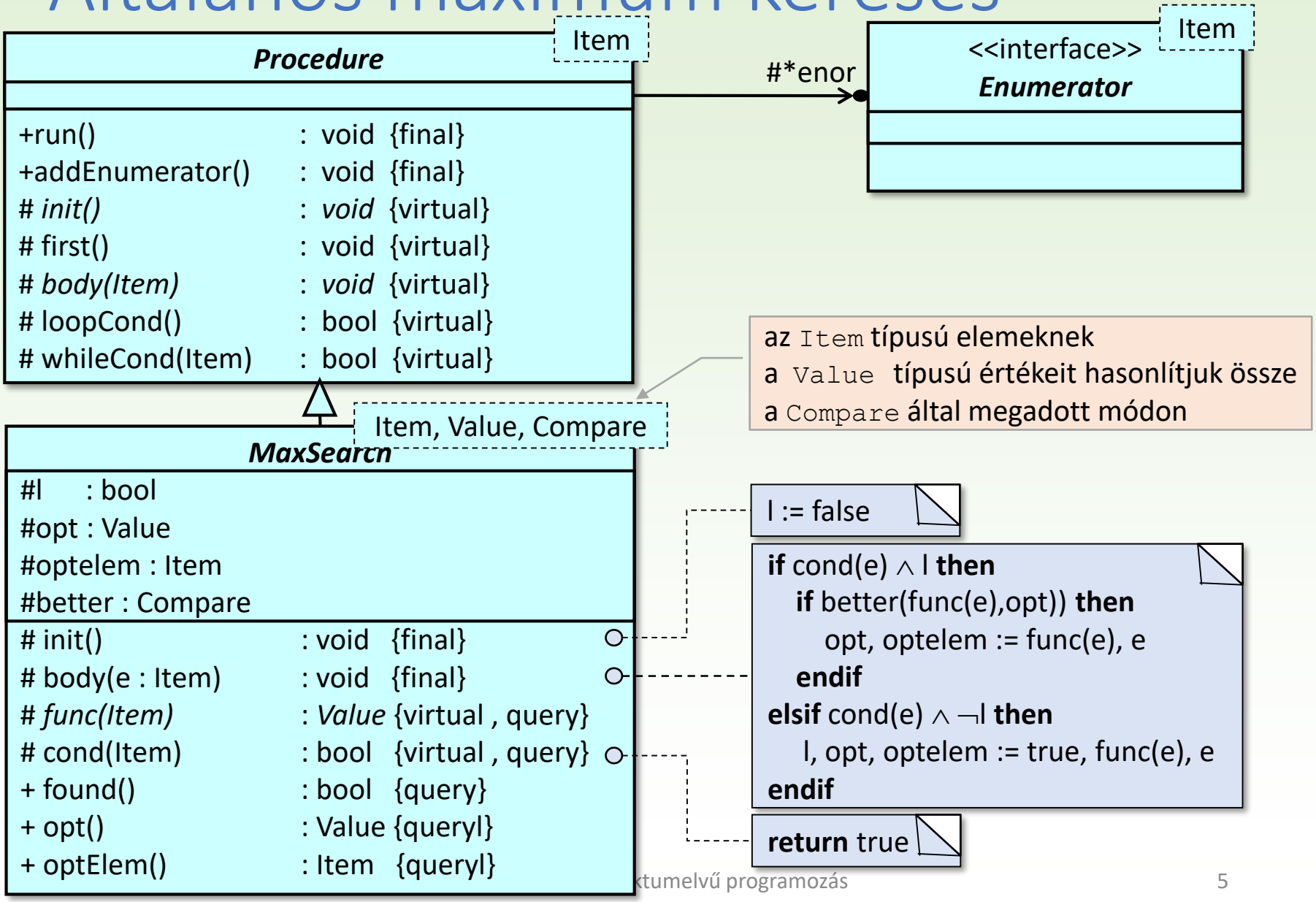
Ez a metódus alkalmas bármelyik programozási tétel algoritmusának végrehajtására, ha az általa hívott (init(), body(), esetenként a loopCond, whileCond) metódusait megfelelő módon felülírjuk, továbbá elérjük, hogy az enor adattag egy alkalmas felsoroló objektumra mutasson.

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(){}
};
```

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;

        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()  const { return _l; }
        Value opt()   const { return _opt; }
        Item optElem()const { return _optelem; }
};
```

összehasonlító adattag

ősciklusban használt metódusok végleges felüldefiniálása

újabb felüldefiniálható metódusok

eredmény lekérdezése

maxsearch.hpp

Maximum keresés ciklusmagja

maximum keresés feltétele

```
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)){
               _opt = val;
               _optelem = e;
          }
     }
     else {
          _l = true;
          _opt = val;
          _optelem = e;
     }
}
```

felsorolt elemekből összehasonlítandó értékeket készít

A `Compare` típusú `_better` objektum mutatja meg, hogy a `val` jobb-e, mint az `_opt`.
Ehhez a `Compare` sablonparamétert helyettesítő típusnak implementálnia kell az operator()-t.

maxsearch.hpp

Összehasonlító osztályok

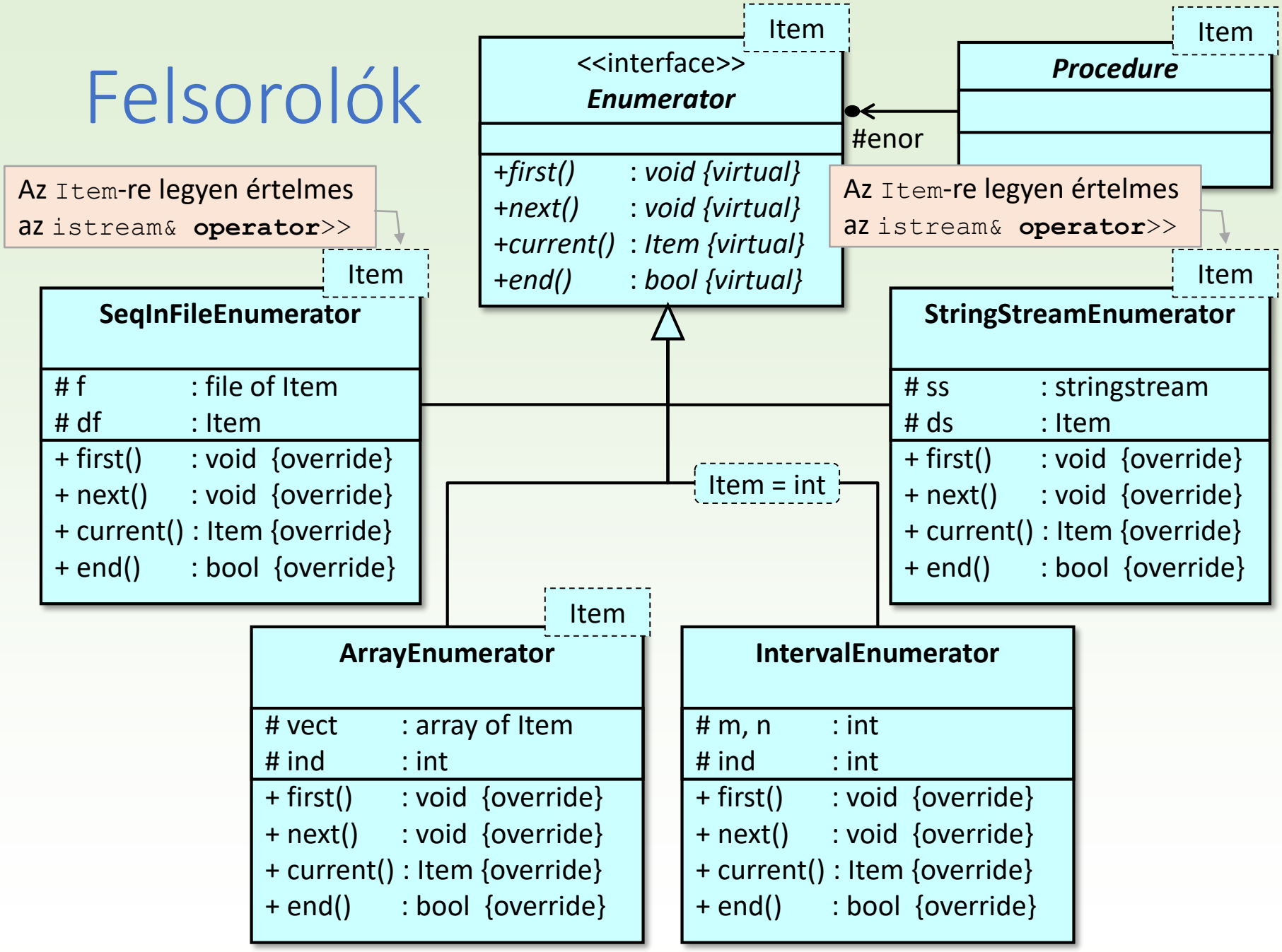
```
template <typename Value>
class Greater{
public:
    bool operator()(const Value& l, const Value& r)
    {
        return l > r;
    }
};

template <typename Value>
class Less{
public:
    bool operator()(const Value& l, const Value& r)
    {
        return l < r;
    }
};
```

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

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

maxsearch.hpp

Felsorolók


Item
<<interface>>
Enumerator
+first() : void {virtual}
+next() : void {virtual}
+current() : Item {virtual}
+end() : bool {virtual}
Item
Procedure
#enor
Az Item-re legyen értelmes
az istream& operator>>
Az Item-re legyen értelmes
az istream& operator>>
Item
SeqInFileEnumerator
f : file of Item
df : Item
+ first() : void {override}
+ next() : void {override}
+ current() : Item {override}
+ end() : bool {override}
Item
StringStreamEnumerator
ss : stringstream
ds : Item
+ first() : void {override}
+ next() : void {override}
+ current() : Item {override}
+ end() : bool {override}
Item = int
Item
ArrayEnumerator
vect : array of Item
ind : int
+ first() : void {override}
+ next() : void {override}
+ current() : Item {override}
+ end() : bool {override}
IntervalEnumerator
m, n : int
ind : int
+ first() : void {override}
+ next() : void {override}
+ current() : Item {override}
+ end() : bool {override}

Intervallum és Tömb felsorolója

intervalenumerator.hpp

```
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; }
};
```

arrayenumerator.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]; }
};
```

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 { next(); }
     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 könyvtárban ez a felsoroló ennél összetettebb: egyrészt rendelkezik egy olyan specializációval, amely karakterenkénti olvasás esetén (Item = char) kikapcsolja az elválasztó jeleket (white space) figyelmen kívül hagyó mechanizmust, másrészt soronkénti olvasás esetén az üres sorokat figyelmen kívül hagyja.

StringStreamEnunerator

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

    void first()          final override { next();}
    void next()           final override { _ss >> _df; }
    bool end()      const final override { return !_ss;}
    Item current() const final 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:infile($\mathbb{N}$) , l:$\mathbb{L}$, max:$\mathbb{N}$

Ef : $f = f_0$

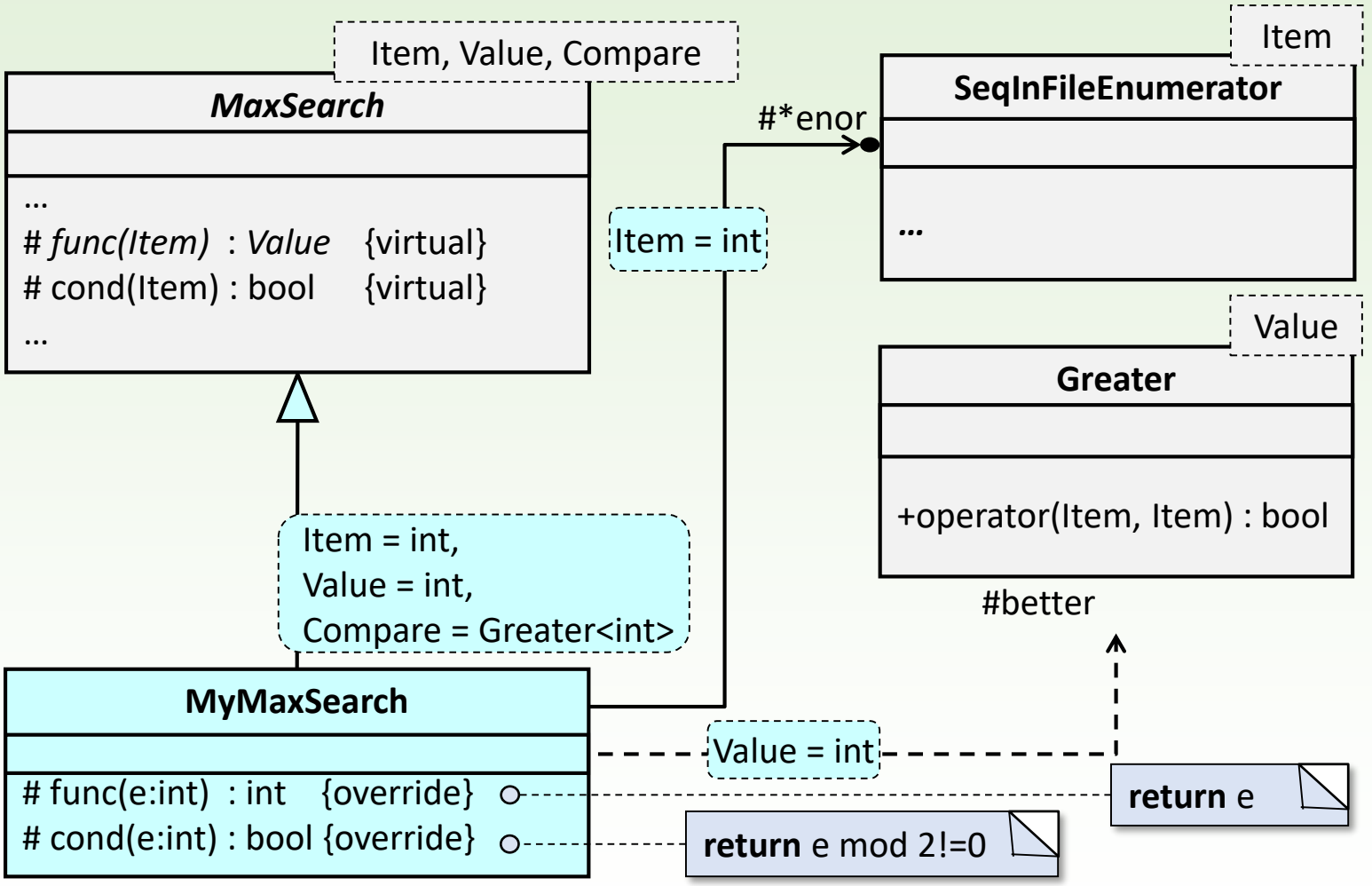
Uf : $l, max = \mathbf{MAX}_{e \in f_0}\ e$
e páratlan

Feltételes maximum keresés

t:enor(E)	~	f:infile($\mathbb{N}$)
f(e)	~	e
H, >	~	$\mathbb{N}$, >
felt(e)	~	e páratlan

E	~	Item
H	~	Value
f(e)	~	func(e)
felt(e)	~	cond(e)

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;}
};
```

```
int main(){
    MyMaxSearch pr;
    SeqInFileEnumerator<int> enor("input.txt");
    // kivételkezelés hiányzik
    pr.addEnumerator(&enor);

    pr.run();

    if (pr.found())
        cout << "The greatest odd integer:" << pr.optElem();
    else
        cout << "There is no odd integer!";
    return 0;
}
```

tevékenység objektum

felsoroló objektum

működés

input.txt

```
12 -5  23
44 130 56 3
-120
```

2. Feladat

Egy forgalom számlálás során egy hónapon keresztül számolták, hogy óránként hány utas lép be egy adott metróállomás területére. (A méréseket hétfőn kezdték, de lehet, hogy nem minden nap és nem minden órában végeztek megfigyelést.)
Az időt négyjegyű számmal kódolták, amelynek első két jegye a mérés napjának a megfigyelés elkezdése óta számolt sorszámát, utolsó két jegye a mérésnek az adott napbeli óráját mutatja. A méréseket egy szöveges állományban rögzítették időkód-létszám párok formájában.
A hétvégi napok közül mikor (melyik nap melyik órájában) léptek be az állomás területére legkevesebben?

input.txt

```
0108 23
0112 44
0116 130
0207 120
…
```

Megoldás specifikációja

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

Ef : $f = f_0$

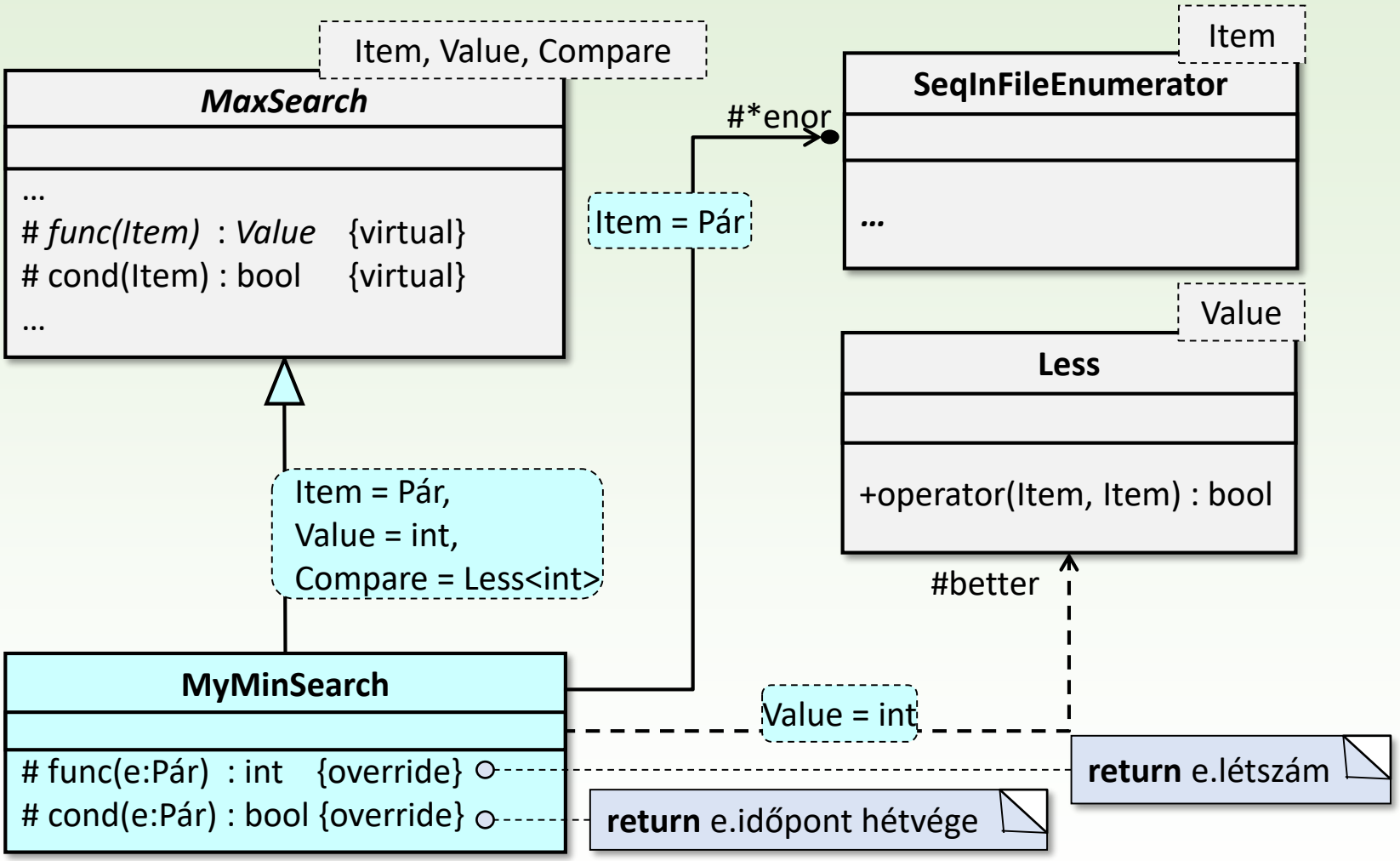
Uf : l, min, elem = $\mathbf{MIN}_{e \in f_0}$ e.létszám
e.időpont hétvége

Feltételes maximum keresés

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

e.időpont/100%7 == 6 || e.időpont/100%7==0

Megoldás osztálydiagramja



Elemi típus (Pár)

input.txt

```
0108 23
0112 44
0116 130
0207 120
…
```

```
struct Pair{
    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, Pair& df)
{
    f >> df.timestamp >> df.number;
    return f;
}
```

Főprogram

```
class MyMinSearch: public MaxSearch<Pair, int, Less<int> > {
    protected:
        int  func(const Pair &e) const override { return e.number; }
        bool cond(const Pair &e) const override
        { return e.day()%7==6 || e.day()%7==0; }
};
```

minimum keresés

létszámok szerint

ezzel a feltétellel

```
int main()
{
    MyMinSearch pr;
    SeqInFileEnumerator<Pair> enor("input.txt");
    // kivételkezelés hiányzik
    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";
    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 recepthez kell cukor?

egy sor elemei alapján

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

Recept = rec(név:String, hozzávalók : Hozzávaló*)

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

Ef : f = f_0

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

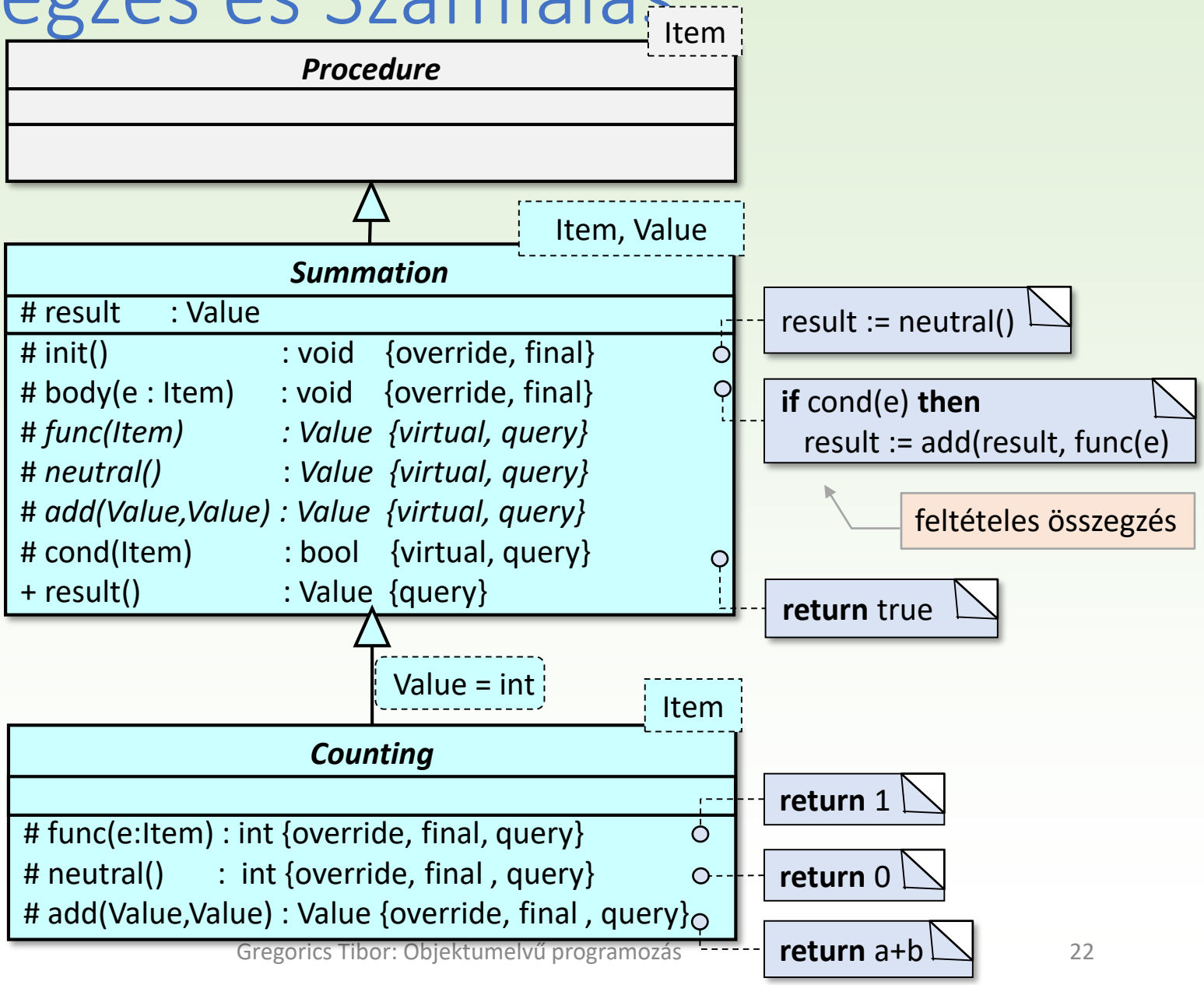
részfeladat: e hozzávalóihoz kell-e cukor?

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

Számlálás		
t:enor(Item)	~	f:infile(Recept)
Item	~	Recept
cond(e)	~	cukros(e)

Lineáris keresés		
t:enor(Item)	~	Hozzávaló* (i=1..*)
Item	~	Hozzávaló
cond(e)	~	e.anyag=cukor

Összegzés és Számlálás



Összegzés és számlálás osztálya

```
template < typename Item, typename Value = Item >
class Summation : public Procedure<Item>{
private:
    Value _result;
protected:
    void init() override final { _result = neutral(); }
    void body(const Item& e) override final {
        if(cond(e)) _result = add(_result, func(e));
    }
    virtual Value func(const Item& e) const = 0;
    virtual Value neutral() const = 0;
    virtual Value add( const Value& a, const Value& b) const = 0;
    virtual bool  cond(const Item& e) const { return true; }
public:
    Value result() const { return _result; }
};
```

summation.hpp

```
template < typename Item >
class Counting : public Summation<Item, int> {
protected:
    int func(const Item& e) const override { return 1; }
    int neutral() const override { return 0; }
    int add( const int& a, const int& b) const override {return a + b;}
};
```

counting.hpp

Sorozatok (ostream, vector) előállítása összegzéssel

A Summation-t felüldefiniáltuk a Value paraméterének speciális eseteire. Ezt használjuk az olyan másolások, listázások, kiválogatások esetén, amikor az output egy sorozat (ostream, vector).

Kívülről megadott listához (ostream) elemek hozzáfűzése adott felsorolásból. A func() metódusnak azt a sztringet kell előállítania, amelyet majd az add() metódus fűz hozzá az ostream típusú eredményhez.

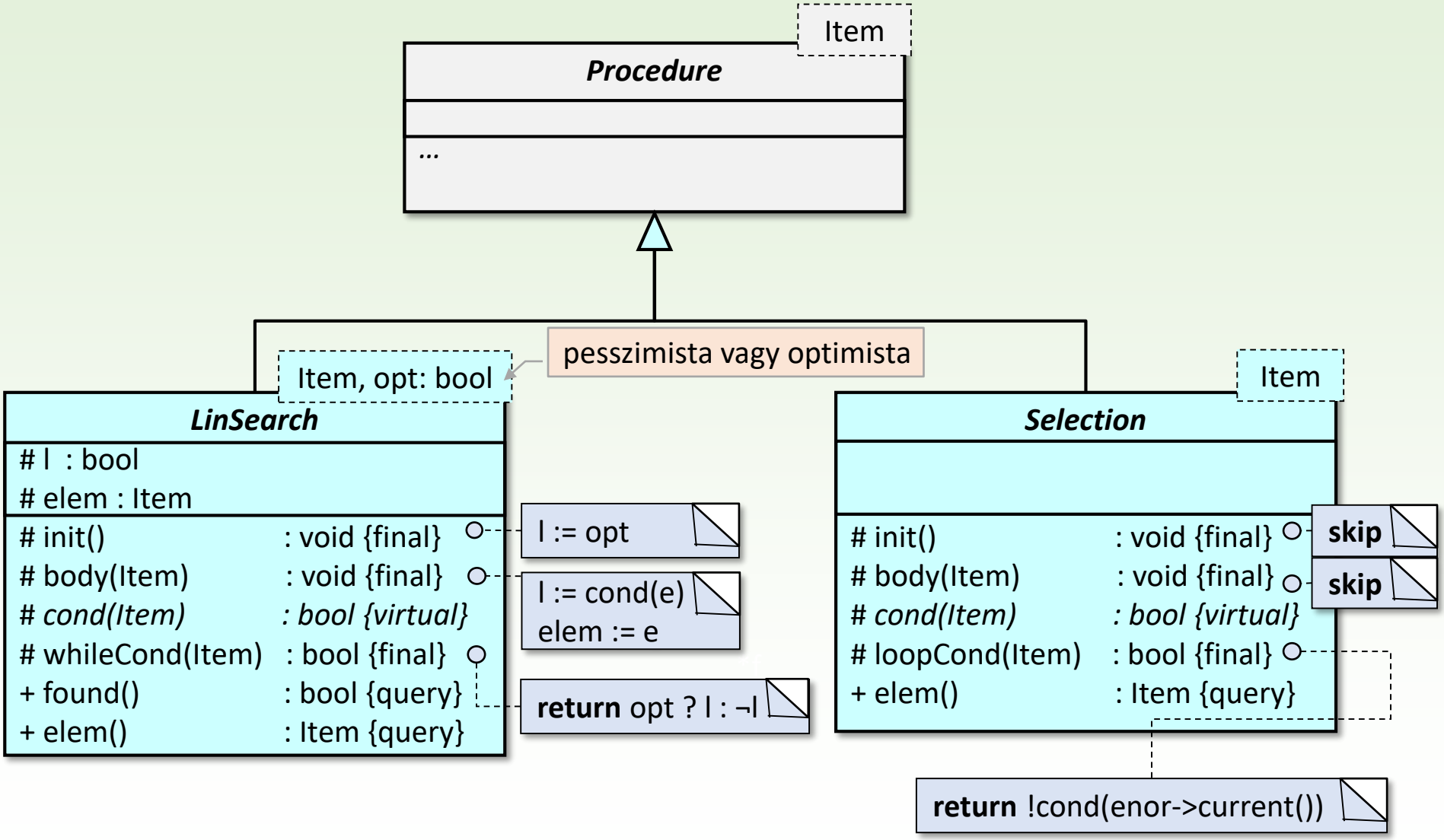
```
class MySum : public Summation<T, ostream>{
public:
    MySum(ostream* o) : Summation<T, ostream>(o) {}
    string func(const T &e) const override {
        ostringstream os;  os << … ;  return os.str();
    }
};
```

Nem kell felülírni neutral() és add()
Felülírható: cond()

Kívülről megadott tömbhöz (vector) elemek hozzáfűzése adott felsorolásból. A func() metódusnak azt az elemet kell előállítania, amelyet majd az add() metódus fűz hozzá az vector<T2> típusú eredményhez.

```
class MySum : public Summation<T1, vector<T2> >{
public:
    MySum(const vector<T2> &v) : Summation<T1, vector<T2> >(v) {}
    T2 func(const T1 &e) const override { return … ; }
};
```

Lineáris keresés és kiválasztás



Lineáris keresés osztálya

```
template < typename Item, bool optimist = false>
class LinSearch : public Procedure<Item> {
protected:
    bool _l;
    Item _elem;

    void init()               override final { _l = optimist; }
    void body(const Item& e) override final { _l = cond(_elem = e); }
    bool whileCond(const Item& e) const override final {
        return optimist?_l:!_l;
    }
    virtual bool cond(const Item& e) const = 0;

public:
    bool found()  const { return _l; }
    Item elem()   const { return _elem; }
};
```

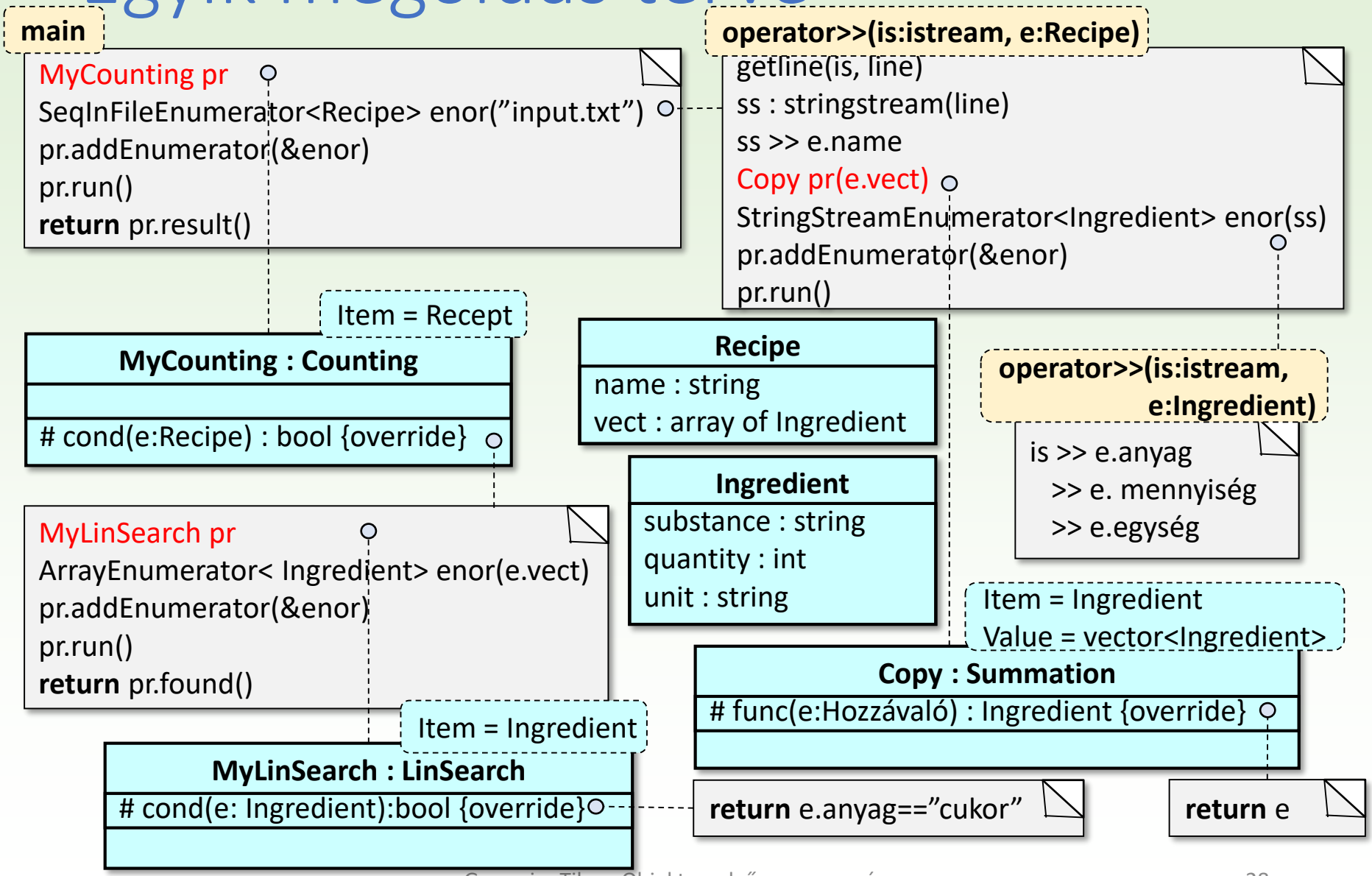
linsearch.hpp

Kitérő: Kiválasztás osztálya

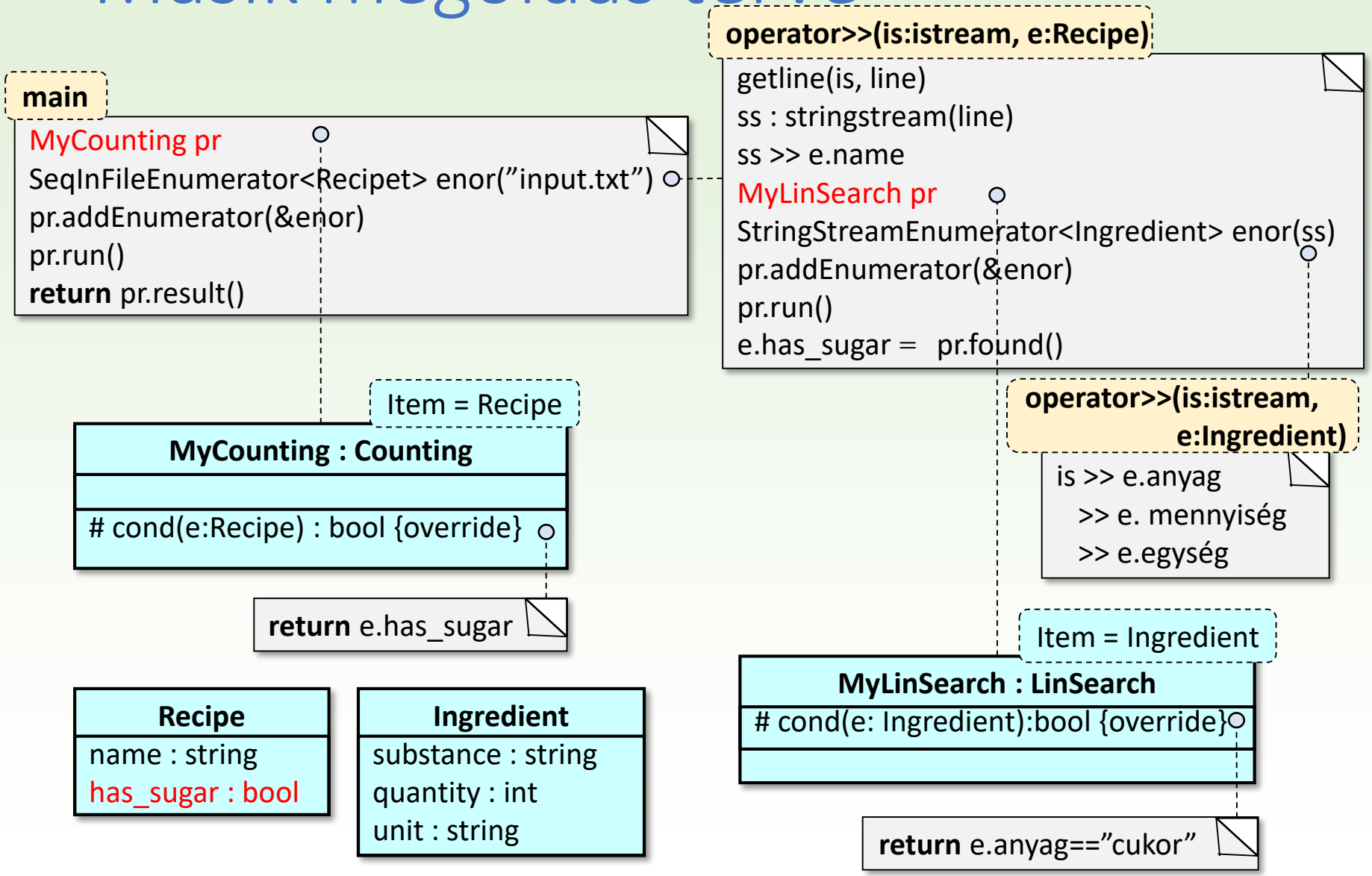
selection.hpp

```
template < typename Item >
class Selection : public Procedure<Item> {
protected:
    void init()               override final {}
    void body(const Item& e) override final {}
    bool loopCond()     const override final {
        return !cond(Procedure<Item>::_enor->current());
    }
    virtual bool cond(const Item& e) const = 0;
public:
    Item result() const { return Procedure<Item>::_enor->current(); }
};
```

Egyik megoldás terve



Másik megoldás terve



4. Feladat

Egy szöveges állományban aszteroidákról vett megfigyeléseket tárolnak. Minden sor egy-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 méréskor 1 milliárd kilométernél közelebb voltak a Földhöz!

A : f:inFile(Megfigyelés) , cout:outfile(String × $\mathbb{N}$)
Megfigyelés = rec(azon:String, dátum:String, tömeg:$\mathbb{N}$, távolság:$\mathbb{N}$)

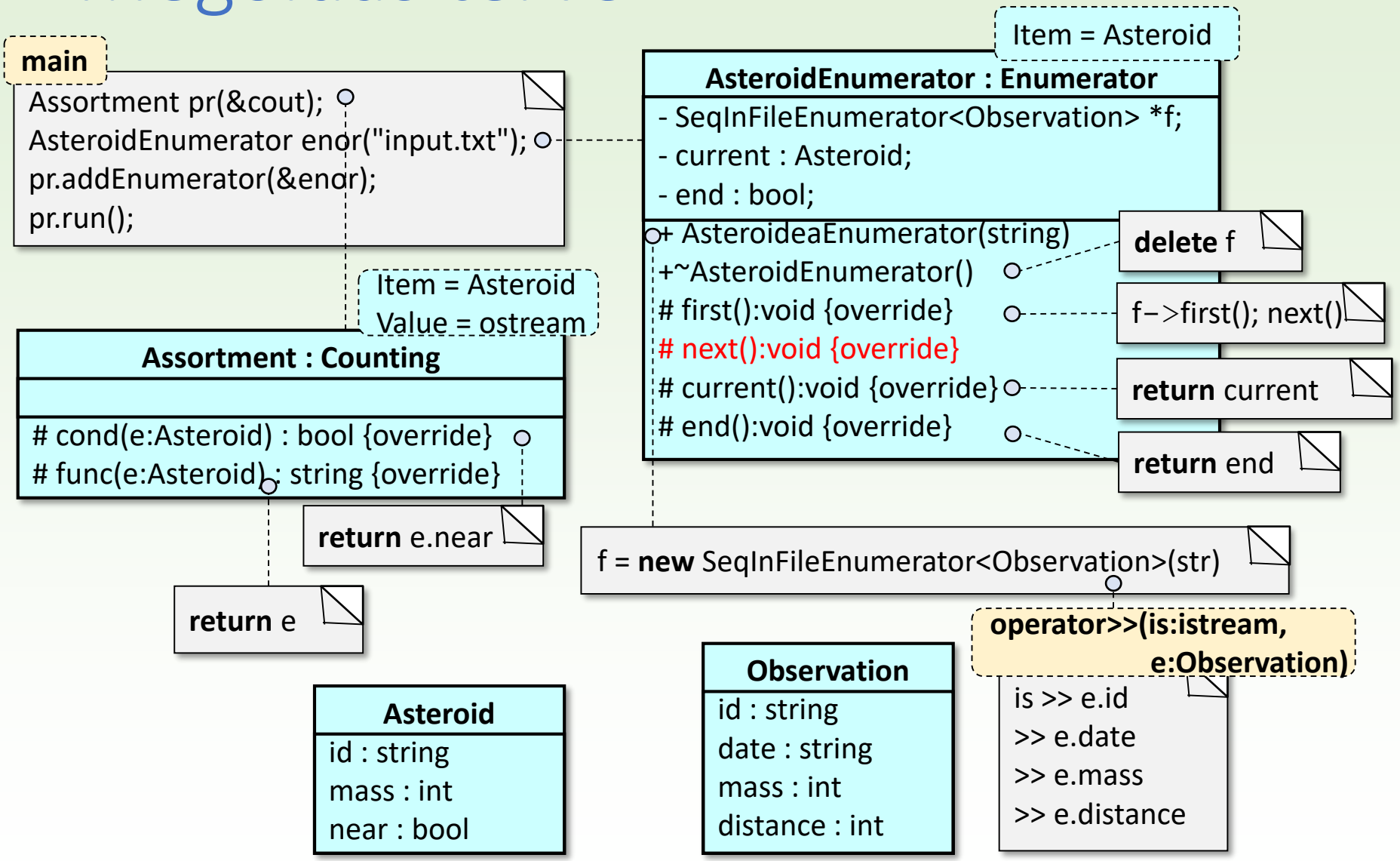
A : t:enor(Aszteroida) , cout:outfile(String × $\mathbb{N}$)
Aszteroida = rec(azon:String, tömeg:$\mathbb{N}$, közel:$\mathbb{L}$)

Ef : $t = t_0$

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

Összegzés
t:enor(Item) ~ t:enor(Aszteroida)
Item ~ Aszteroida
func(e) ~ (e.azon, e.tomeg)
Value, +, 0 ~ (String × $\mathbb{N}$)*, $\oplus$, <>)
cond(e) ~ e.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:inFile(Megfigyelés), e:Megfigyelés, st:Status, akt:Aszteroida, vége:$\mathbb{L}$

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

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

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

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

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

ezt a két felsorolást nem lehet szekvenciában végezni, össze kell vonni egy ciklusba

ez a két feldolgozás eltérő állapotokban állhat le

Két programozási tétel összevonása

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

nem vonható össze

Maximum kiválasztás

t:enor(Item)	~	f:infile(Megfigyelés) first() nélkül amíg azonosítót nem vált
Item	~	Megfigyelés
func(e)	~	e.tömeg
Value, >	~	$\mathbb{N}$, >

Optimista lineáris keresés

t:enor(Item)	~	f:infile(Megfigyelés) first() nélkül amíg azonosítót nem vált
Item	~	Megfigyelés
cond(e)	~	e.táv < 10000

Összegzés

t:enor(Item)	~	f:infile(Megfigyelés) first() nélkül amíg azonosítót nem vál
Item	~	Megfigyelés
func(e)	~	e.tömeg
Value, +, 0	~	$\mathbb{N}$, max, 0

összevonható

Összegzés

t:enor(Item)	~	f:infile(Megfigyelés) first() nélkül amíg azonosítót nem vált
Item	~	Megfigyelés
func(e)	~	e.táv < 10000
Value, +, 0	~	$\mathbb{L}$, $\wedge$, igaz

next() metódus terve