Objektumelvű programozás

Gregorics Tibor

gt@inf.elte.hu

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

Procedurális vs. objektumelvű programozás

- Procedurális programozás: egy probléma részfeladatait megoldó tevékenységeket önálló egységekbe, ún. procedurákba (részprogram, makró, eljárás, függvény) szervezzük. A problémát megoldó folyamatot ezen procedurák közötti vezérlés-átadások (eljárások, függvények esetében hívások) láncolata mutatja meg.
- Objektumelvű programozás: egy probléma megoldáshoz szükséges adatok egy-egy részét a hozzájuk kapcsolódó tevékenységekkel (az ún. metódusokkal) együtt egységekbe, ún. objektumokba zárjuk. A problémát megoldó folyamatot ezen objektumok metódusai közötti vezérlés-átadások (hívások vagy szignálok) jelöli ki.

Feladat

Egy számsorozatban 0 és m közé eső természetes számok találhatók. Melyik a sorozat leggyakoribb eleme?

- Procedurális megoldás: maximum kiválasztás és számlálás
  - Megszámoljuk a számsorozat elemeinek a számsorozatbeli gyakoriságát, és ezek közül megkeressük a legnagyobbat.
- Objektumelvű megoldás: tároló objektum
  - Elhelyezzük a tároló objektumban a számsorozat elemeit, majd lekérdezzük a benne található leggyakoribb elemet. Legyen az elhelyezés is, és a lekérdezés is gyors.

Elemzés

Ezek a feladat (input, output) változói, amelyek a megoldást adó program változói is lesznek.

$A = (\ m{:}\mathbb{N}\ ,\ x{:}\mathbb{N}^*,\ elem{:}\mathbb{N}\ )$

m kezdőértéke: $m_0 \in \mathbb{N}$
x kezdőértéke: $x_0 \in \mathbb{N}^*$

x_0 legalább 1 hosszú sorozat, x_0 elemei 0 és m_0 közé esnek

$Ef = (\ m = m_0 \wedge x = x_0 \ \wedge\ |x| \geq 1 \ \wedge\ \forall i \in [1\ ..\ |x|]: x[i] \in [0\ ..\ m]\ )$

A változók értékét jellemzi, még a megoldó program elindulása előtt.

Dobáljuk be x_0 elemeit a zsákba, majd kérjük le annak a leggyakoribb elemét.

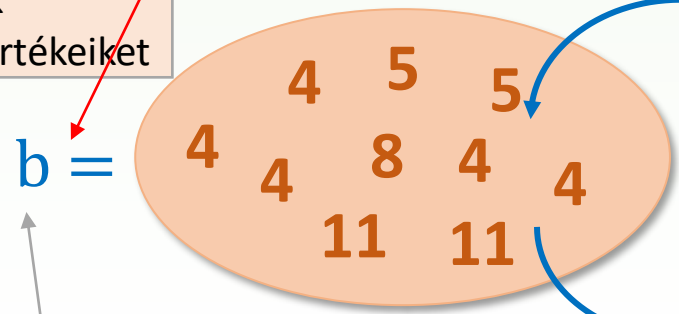
$Uf = (\ m = m_0 \wedge x = x_0 \ \wedge\ b{:}Zsák \ \wedge\ b = \bigcup_{i=1}^{|x|} [x[i]] \ \wedge\ elem = leggyakoribb(b)\ )$

A változók értékét jellemzi, már a megoldó program leállása után.

az input-változók megőrzik kezdőértékeiket

b:=b∪[e]

Itt az ∪ egy zsákot egy zsákkal egyesítő homogén binér művelet, amelynek neutrális eleme az üres zsák (∅).
Az [e] pedig az e-t tartalmazó egyelemű zsákot jelöli.



Zsák típusú segédváltozó

e:=leggyakoribb(b)

Tervezés

$A = (m:\mathbb{N} , x:\mathbb{N}^*, elem:\mathbb{N})$

$Ef = (m = m_0 \wedge x = x_0 \quad \wedge \quad |x| \geq 1 \quad \wedge \quad \forall i \in [1 .. |x|]: x[i] \in [0 .. m])$

Dobáljuk be x_0 elemeit a zsákba, majd kérjük le a zsák leggyakoribb elemét.

$Uf = (m = m_0 \wedge x = x_0 \quad \wedge \quad b:Zsák \quad \wedge \quad b = \bigcup_{i=1}^{|x|} [x[i]] \quad \wedge \quad elem = leggyakoribb(b))$

végrehajtható specifikáció: elmosódik az elemzés és a tervezés közötti határ (mi a feladat és hogyan oldjuk meg)

A bezsákolás egy speciális összegzés:

$$b = \sum_{i=1}^{|x|} x[i] \quad \sim \quad b = \bigcup_{i=1}^{|x|} [x[i]]$$

elem: $x[i] \sim [x[i]]$

művelet: $\mathbb{N}, +, 0 \sim$ Zsák, $\bar{\cup}$, $\emptyset$

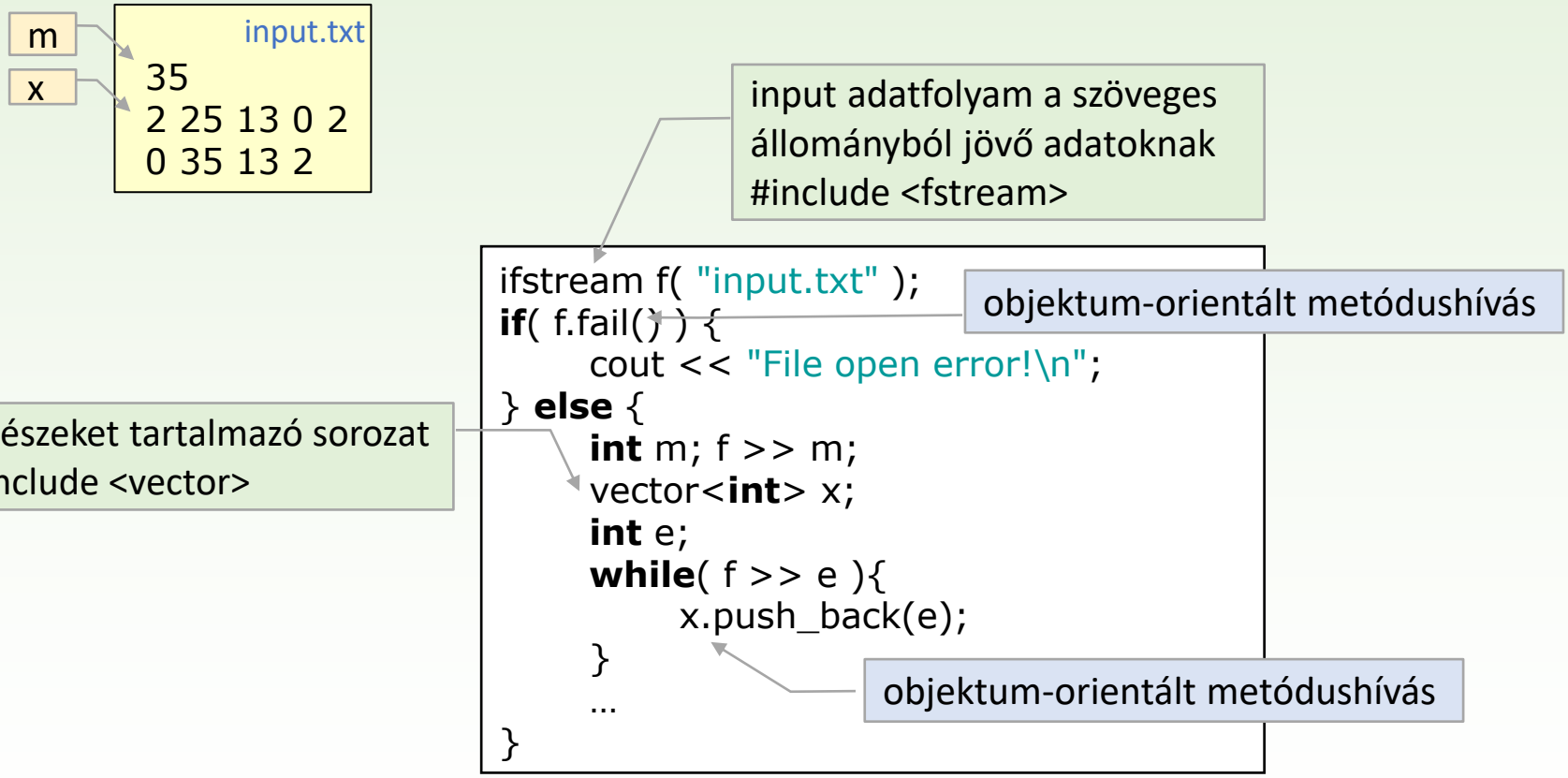
```
b := ∅
i = 1 .. |x|
    b := b ∪̄ [ x[i] ]
elem := leggyakoribb(b)
```

Összegzés általánosítása:

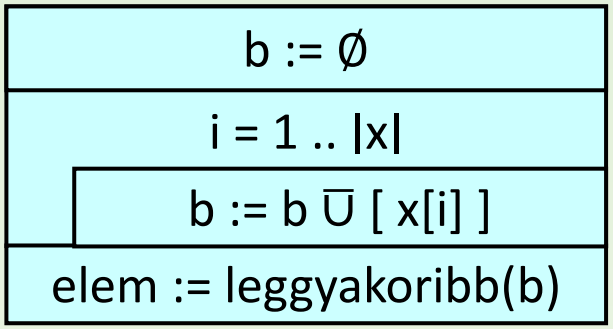
$s = \sum_{i=m..n} f(i)$, ahol $f:[m..n] \longrightarrow H$, és a $+:H \times H \longrightarrow H$ művelet neutrális eleme a 0.

Megvalósítás: kitérő

Hogyan töltünk fel C++ programozási nyelven egy sorozatot egy szöveges állomány adataival?



Megvalósítás



m
x

input.txt
35
2 25 13 0 2
0 35 13 2

```
int main()
{
	ifstream f( "input.txt" );
	if( f.fail() ){
		cout << "File open error!\n"; return 1;
	}
	int m; f >> m;
	Bag b(m);
	b.erase();
	int e;
	while( f >> e ){
		b.putIn(e);
	}
	cout << "The most frequent number: " << b.mostFrequent() << endl;

	return 0;
}
```

Zsák

b := Ø

Nem építjük fel az x sorozatot csak azért,
hogy majd abból feltöltsük a zsákot.
Ehelyett közvetlenül a fájl elemeit pakoljuk a zsákba.

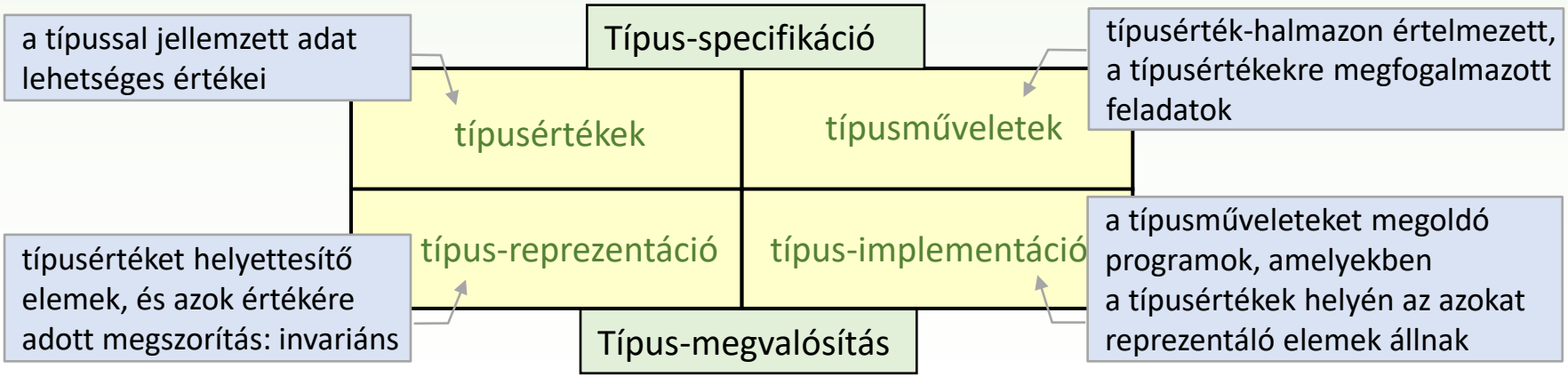
b := b ∪ [e]

leggyakoribb(b)

Hogyan adjuk meg a Bag és
a műveleteinek a jelentését?
Hátra van még a Zsák típus
tervezése és kódolása.

Adattípus fogalma

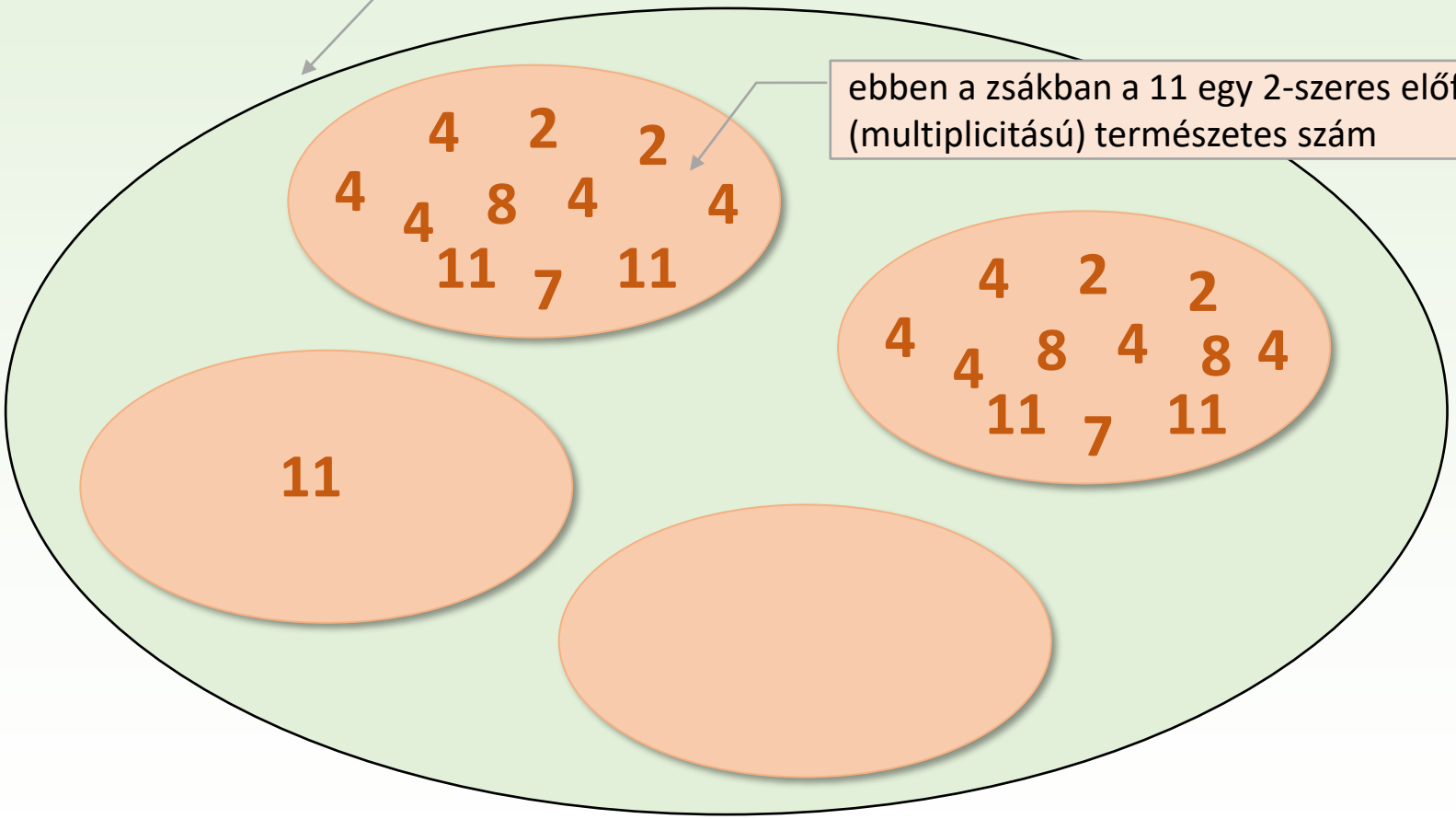
- Egy adat (változó) típusának definiálásához szükség van a típus specifikációjára és annak megvalósítására.
- A típus-specifikáció megadja:
  - az adat által felvehető értékek halmazát
  - a típusértékekkel végezhető műveleteket
- A típus-megvalósítás megmutatja:
  - hogyan ábrázoljuk (reprezentáljuk) a típus értékeit
  - milyen programok helyettesítsék (implementálják) a műveleteket



Zsák típus típusérték-halmaza

Egy természetes számokat tároló zsák típusú adatnak (változónak) az értékei (típusértékei) természetes számokat tároló zsákok. Az összes ilyen zsák alkotja a zsák típus típusérték-halmazát.

ebben a zsákban a 11 egy 2-szeres előfordulású (multiplicitású) természetes szám



Zsák típus műveletei

Kiüríti a zsákot:

$b := \emptyset$ $\quad b{:}\text{Zsák}$

adjon hibajelzést, ha $e \notin [0 \,..\, m]$

Betesz egy elemet a zsákba:

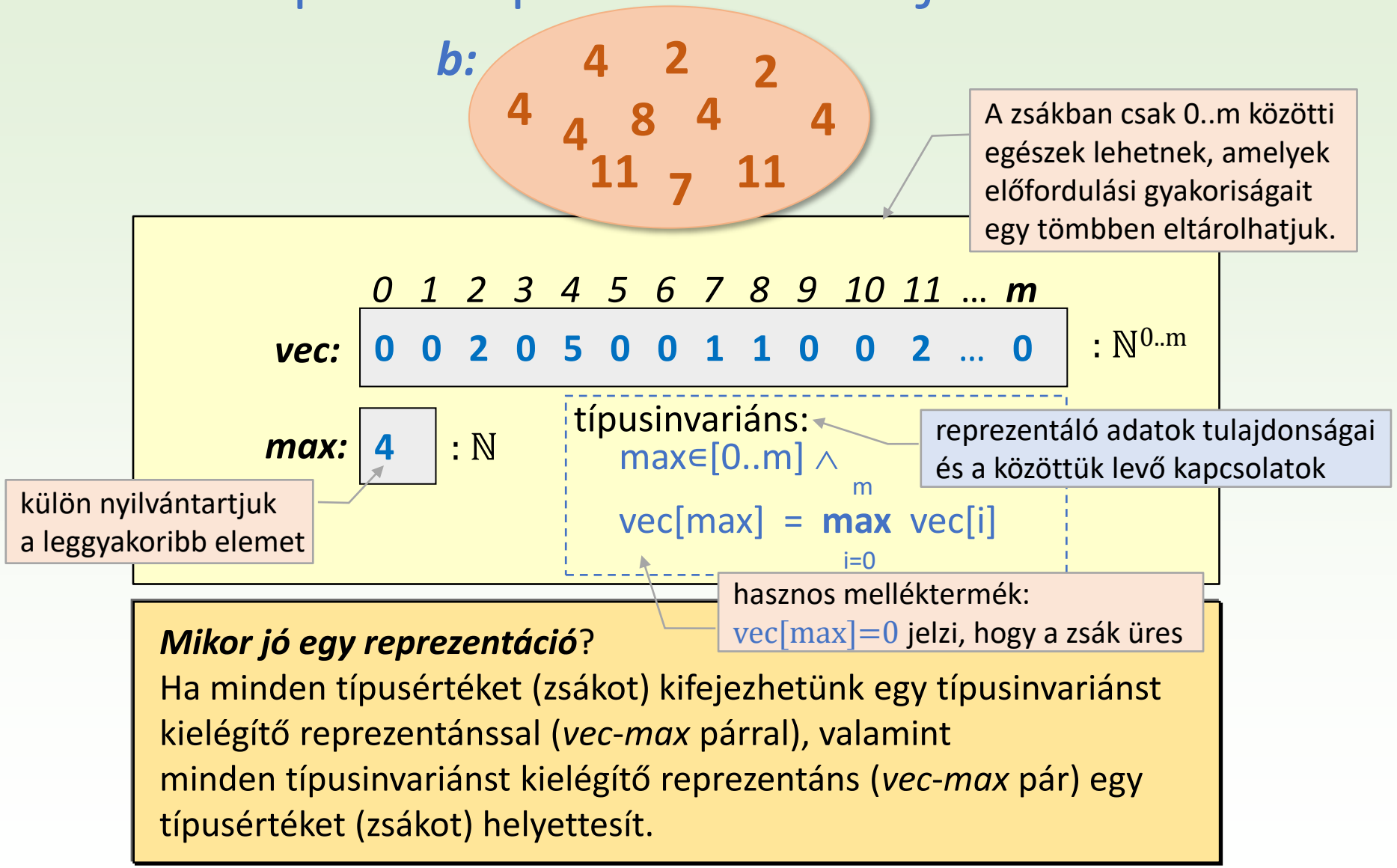
$b := b \cup [e]$ $\quad b{:}\text{Zsák}, e{:}\mathbb{N}$

adjon hibajelzést, ha a zsák üres

Zsák leggyakoribb eleme:

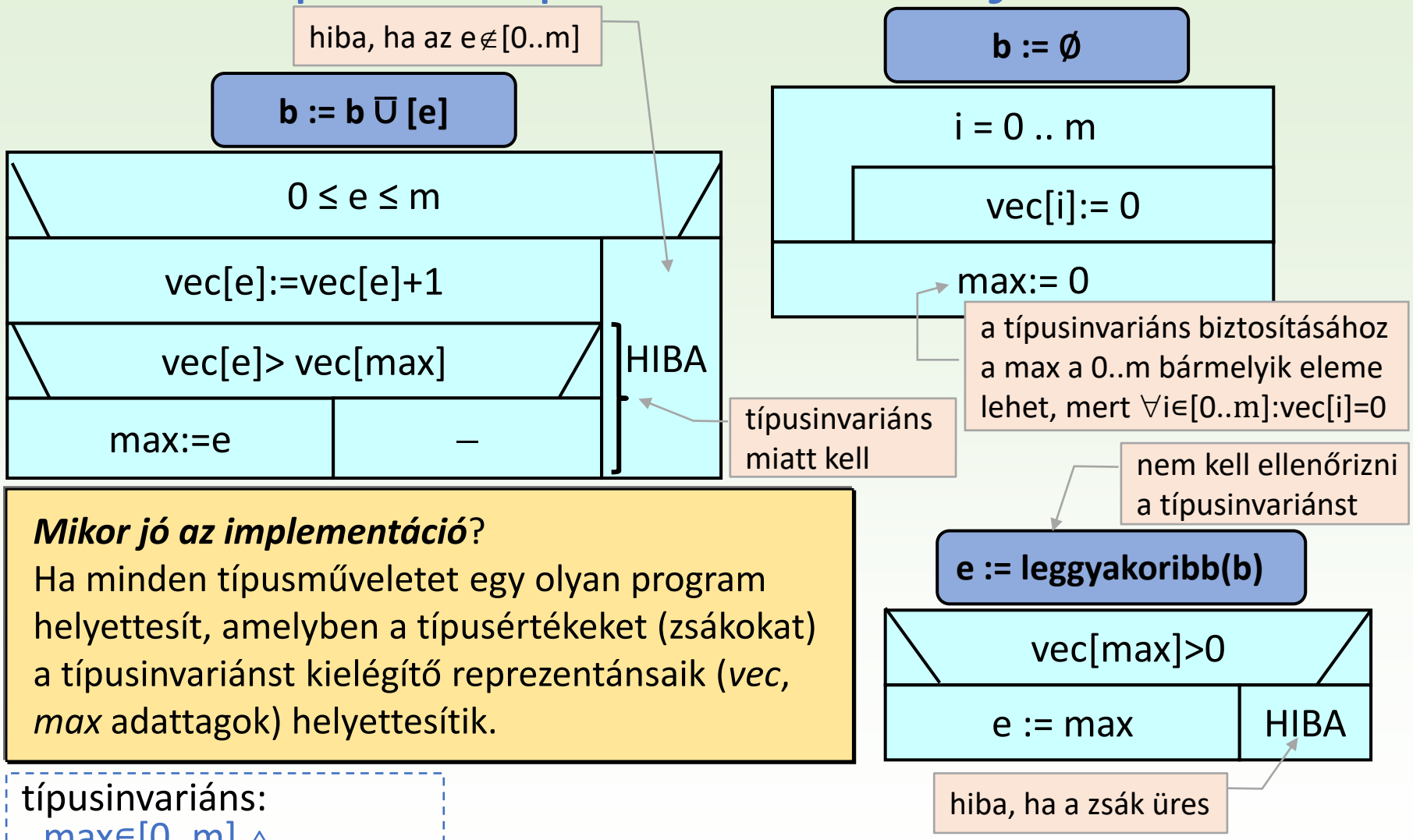
$e{:=}\text{leggyakoribb}(b)$ $\quad b{:}\text{Zsák}, e{:}\mathbb{N}$

Zsák típus reprezentációja



Mikor jó egy reprezentáció?
Ha minden típusértéket (zsákot) kifejezhetünk egy típusinvariánst kielégítő reprezentánssal (*vec-max* párral), valamint
minden típusinvariánst kielégítő reprezentáns (*vec-max* pár) egy típusértéket (zsákot) helyettesít.

Zsák típus implementációja



Mikor jó az implementáció?
Ha minden típusműveletet egy olyan program helyettesít, amelyben a típusértékeket (zsákokat) a típusinvariánst kielégítő reprezentánsaik (*vec*, *max* adattagok) helyettesítik.

típusinvariáns:

$$max \in [0..m] \wedge$$

$$vec[max] = \max_{i=0}^{m} vec[i]$$

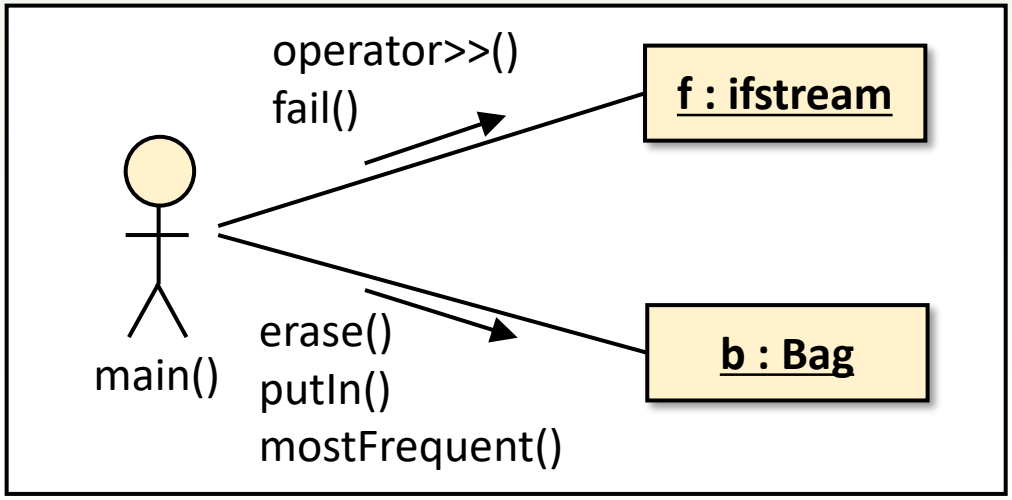
Objektum fogalma

például egy zsák (Bag)

- Egy objektum a feladat megoldásának adott részéért felelős egység, amely tartalmazza az ezen részhez tartozó adatokat és az ezekkel kapcsolatos műveleteket.

vec, max

erase(), putIn(), mostFrequent()



Osztály fogalma

- Az osztály egy objektum szerkezetének és viselkedésének a mintáját adja meg, azaz
  - felsorolja az objektum adattagjait a nevük és típusuk megadásával
  - megadja az objektumra meghívható metódusokat (tagfüggvény, művelet) nevük, paraméterlistájuk, visszatérési értékük típusa, és törzsük megadásával
- Az osztály lényegében az objektum típusa: az objektumot az osztálya alapján hozzuk létre, azaz példányosítjuk.
- Egy osztályhoz több objektum is példányosítható: minden objektum rendelkezik az osztályleírás által leírt adattagokkal és metódusokkal.

vec : int[0..m], max : int

erase() : void
putIn(int) : void
mostFrequent() : int

Osztály UML jelölése

- Egy osztály rendelkezik
  - névvel,
  - adattagokkal, (tulajdonság, attribútum, mező)
  - metódusokkal
- Az adattagok és metódusok láthatósága egyenként beállítható
  - kívülről is látható, azaz publikus (*public* +)
  - külvilág elől rejtett: privát (*private* -) vagy védett (*protected* #)

<osztálynév>
<+|-|#> <adattagnév> : <típus> …
<+|-|#> <metódusnév>(<paraméterek>) : <típus> …

Bag
- vec : int[0..m] - max : int
+ erase() : void + putIn(e:int) : void + mostFrequent() : int

Típus és Osztály

- Az objektumelvű tervezés során osztályként adjuk meg az egyedi, vagy ún. felhasználói típusokat (*custom type*).

Zsák típus:

típus-specifikáció

típus-értékek

típus-műveletek

zsákok

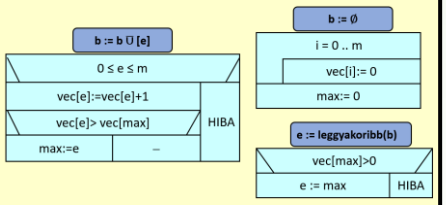
b := ∅
b := b ∪ [e]
e:=leggyakoribb(b)
b:Zsák, e:ℕ

reprezentáció

implementáció

vec : $\mathbb{N}^{0..m}$
max : ℕ

{ max∈[0..m] ∧
vec[max] =
max$_{i=0..m}$ vec[i] }



típus-megvalósítás

Zsák osztály:

max∈[0..m] ∧
vec[max] = **max**$_{i=0..m}$ vec[i]

Bag

- vec : int[0..m]
- max : int

+ erase() : void
+ putIn(e:int) : void
+ mostFrequent() : int

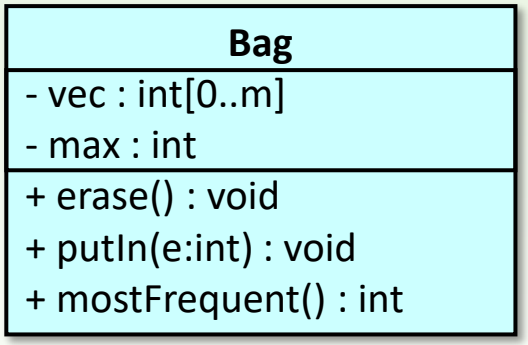
```
for i=0..m loop vec[i] := 0 endloop
max := 0
```

```
if vec[max]>0 then return max
else error
endif
```

```
if 0≤e≤m then
    ++vec[i]
    if vec[e]>vec[max] then  max := e endif
else error
endif
```

Zsák típus osztálya C++ nyelven

- A C++ nyelvben az osztályt a *class* nyelvi elem segítségével írjuk le.



típus neve

```
class Bag{

private:
        vector<int> _vec;
        int _max;

public:
        void erase();
        void putIn(int e);
        int  mostFrequent() const;
};
```

adattagok

metódusok

Ez egy konstans metódus, amely nem változtatja meg az objektum adattagjainak értékét.

1

Az objektum-orientált nyelvek lényeges ismérve az egységbezárás: egy adott feladatkör megvalósításához szükséges adatokat és az azokat manipuláló programrészeket a program többi részétől elkülönítve adhatjuk meg.

Hivatkozás az objektum tagjaira

```
class Bag{
private:
    vector<int> _vec;
    int _max;
public:
    void erase();
    void putIn(int e);
    int  mostFrequent() const;
};

int main()
{
    Bag b1, b2;
    b1.erase();
    b2.putIn(5);
    int a = b2.mostFrequent();
    b1._max = 0;
    b1._vec[5]++;
}
```

objektumok példányosítása

Amikor egy objektum egy tagjára (adattagra vagy metódusra) hivatkozunk, akkor magát az objektumot (pontosabban az arra hivatkozó változót) odaírjuk a tag elé.

A metódus elé írt objektum a metódus kitüntetett, extra paramétere. (Konstans metódus esetén ez egy csak bemenő paraméter.)

Egy objektum rejtett (privát, védett) tagjait csak az objektum metódusainak definíciójában (azok törzsében) használhatjuk, máshol ezekre nem hivatkozhatunk közvetlenül.

2

Az objektum orientált nyelvek fontos ismérve az elrejtés: az egységbe zárt elemek láthatóságát korlátozhatjuk. (Általában az adattagok rejtettek, azok értékéhez csak közvetetten, a publikus metódusokkal férünk hozzá.)

Zsák típus metódusai C++ nyelven

a Bag osztályhoz tartozó

sorozat hossza: unsigned int típusú

```
void Bag::erase() {
      for(unsigned int i=0; i<_vec.size(); ++i) _vec[i] = 0;
      _max = 0;
}

void Bag::putIn(int e) {
      if( e<0 || e>=int(_vec.size()) ) return; // hibakezelés még hiányzik
      if( ++_vec[e] > _vec[_max] ) _max = e;
}

int Bag::mostFrequent() const {
      if( 0 ==_vec[_max] ) return -1; // hibakezelés még hiányzik
      return _max;
}
```

castolás int-re

Az olyan tagok, amelyek előtt nincs feltüntetve, hogy melyik objektumhoz tartoznak, alapértelmezés szerint annak az objektumnak a tagjai, amelyikkel az őket tartalmazó a metódust meghívják.

Konstruktor

Egy objektum példányosításakor az osztálynak egy speciális metódusa, a konstruktora hívódik meg, amelyik sorban egymás után hozza létre az objektum adattagjait (azok konstruktoraival).

A példányosításhoz – amíg mást konstruktort nem definiálunk – alapértelmezetten rendelkezésünkre áll egy üres konstruktor. Ennek nincs paramétere, és az objektum adattagjait azok üres konstruktoraival hozza létre (a fordító jelzi, ha nincs nekik ilyen).

nincs visszatérési típusa
neve: az osztályának neve

```
class Bag{
private:
    vector<int> _vec;
    int _max;
public:
    Bag(int m);
    …
};
```

A *Bag b* deklaráció az üres konstruktort hívja, amely létrehoz egy nulla hosszú _vec nevű sorozatot, és egy _max nevű integert.

Kell egy olyan paraméteres konstruktor, amely beállítja a _vec hosszát (pl.: *Bag b(35)*) és lefuttatja az erase() metódust.

Létrehozza az adattagokat azok üres konstruktorainak implicit módon történő hívásaival, majd végrehajtja a törzse utasításait.

Alternatívák a konstruktorra:

```
Bag::Bag(int m) { _vec.resize(m+1); erase(); }
```

```
Bag::Bag(int m) { _vec.resize(m+1, 0); _max = 0; }
```

A vector átméretezését végző metódusnak több alakja van: megadható a sorozatot feltöltő érték.

Itt explicit módon hívjuk meg az adattagok konstruktorait.

```
Bag::Bag(int m) : _vec(m+1, 0), _max(0) { }
```

Zsák osztály önálló állományban

fejállományok elejére, hogy kizárjuk a többszörös bemásolásokat (nem szabványos)

```
#pragma once

#include <vector>

class Bag{
private:
	std::vector<int> _vec;
	int _max;
public:
	Bag(int m);
	void erase();
	void putIn(int e);
	int  mostFrequent() const;
};
```

A fejállomány elejére nem szokás beírni a *using namespace std*-t, ezért külön jelezzük, hogy a vector definíciója az std névtérben található.

bag.h

Külön fejállomány (header fájl), amelyet minden olyan forrásfájlba „beinklúdolunk”, ahol a benne levő definíciókat használni akarjuk.

Metódusok külön fordítási egységben

```
#include "bag.h"

Bag::Bag(int m) : _vec(m+1, 0), _max(0) { }

void Bag::erase() {
      for(unsigned int i=0; i<_vec.size(); ++i) _vec[i] = 0;
      _max = 0;
}
void Bag::putIn(int e) {
      if( e<0 || e>=int(_vec.size()) ) {
            cout << "Wrong input: " << e << endl;
            return;
      }
      if( ++_vec[e] > _vec[_max] ) _max = e;
}
int Bag::mostFrequent() const {
      if( 0 ==_vec[_max] ) {
            cout << "No most frequented element!" << endl;
            return -1;
      }
      return _max;
}
```

Bag osztály definíciójának bemásolása

Az ilyen közvetlen hibajelzés helyett később majd kivételkezelést fogunk használni.

külön fordítási egység

bag.cpp

Főprogram

Bag osztály definíció bemásolása

projekt:
- main.cpp
- bag.h
- bag.cpp

```
#include <iostream>
#include <fstream>
#include "bag.h"
using namespace std;

int main()
{
	ifstream f( "input.txt" );
	if( f.fail() ){
		cout << "File open error!\n";
		return 1;
	}
	int m; f >> m;
	if( m<0 ){
		cout << "Upper limit of natural numbers cannot be negative!\n";
		return 1;
	}
	Bag b(m);
	int e;
	while( f >> e ) {
		b.putIn(e);
	}
	cout << "The most frequent element: "<< b.mostFrequent() << endl;
	return 0;
}
```

main.cpp

https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md

Automatikus tesztelés

```
#include <iostream>
#include <fstream>
#include "bag.h"
using namespace std;


#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("empty sequence", "[main]"){
        ifstream f( "input1.txt" ); REQUIRE(!f.fail());
        int m; f >> m; CHECK(m==35); REQUIRE(m>=0);
        Bag b(m);
        int e;
        while( f >> e) { b.putIn(e); }
        CHECK(b.mostFrequent()==-1);
}
TEST_CASE("one-length sequence", "[main]"){
        ifstream f( "input2.txt" ); REQUIRE(!f.fail());
        int m; f >> m; CHECK(m==35); REQUIRE(m>=0);
        Bag b(m);
        int e;
        while( f >> e ){ b.putIn(e); }
        CHECK(b.mostFrequent()==2);
}
...
```

input1.txt = < 35 > (m = 35)
b = [ ] → error: No most frequented element

input2.txt = < 35, 2 > (m = 35)
b = [2(1)] → The most frequented element = 2

main.cpp

https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md

Automatikus egység (unit) tesztek

```
TEST_CASE("creation of an empty bag", "[bag]")
{
      int m = 0;
      Bag b(m);
      vector<int> v = { 0 };
      CHECK( v == b.getArray() );
}
TEST_CASE("new element into empty bag", "[putIn]")
{
      Bag b(1);
      b.putIn(0);
      vector<int> v = { 1, 0 };
      CHECK( v == b.getArray() );
}
```

üres zsák létrehozása:
m = 0 → _vec = < 0 >

Mivel a b._vec privát, azt nem tudjuk közvetlenül lekérdezni.

üres zsákba új elem:
m = 1, e = 0 → _vec = < 1, 0 >

```
class Bag {
private:
      std::vector<int> _vec;
      int _max;
public:
      …
      const std::vector<int>& getArray() const {return _vec;}
};
```

Elegánsabb lenne egy Bag_Test osztályt készíteni, amely átvenné (örökölné) a Bag osztály minden tulajdonságát, de kiegészítené azokat a getArray() metódussal, és a tesztelést erre a Bag_Test osztályra végeznénk. Ahhoz azonban, hogy a Bag_Test osztályban a _vec adattagot elérjük, annak láthatóságát a Bag osztályban private-ról protected-re kell módosítani.

„inline” definíció