

PROGRAMOZÁS

*Procedurális vs. objektum orientált
programozás*

Gregorics Tibor

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

Feladat

Melyik a leggyakoribb elem egy sorozatban?

0. változat

*Melyik a leggyakoribb elem egy 0 és m közé eső
egész számokból álló sorozatban?*

Procedurális programterv

$$A = (x:\mathbb{Z}^n, m:\mathbb{N}, e:\mathbb{Z})$$

$$Ef = (x = x' \wedge n > 0 \wedge m = m' \wedge \forall i \in [1..n]: x[i] \in [0..m])$$

$$Uf = (x = x' \wedge (\max, e) = \max_{i=0}^m \text{hány}(i))$$

ahol $\text{hány}(i) = \sum_{j=1}^n 1_{x[j]=i}$

maximum kiválasztásban
számlálás

$\max, e := \text{hány}(0), 0$

$i = 1 .. m$

$c := \text{hány}(i)$

$\max < c$

$\max, e := c, i$

—

$c := \text{hány}(i)$

$s := 0$

$j = 1 .. n$

$x[j] = i$

$s := s + 1$

—

Fekete doboz tesztelés vázlat

Maximum kiválasztás tesztje: (0 .. m)

<u>intervallum</u> hossza:	$m = 0,$	$x = [\dots] \rightarrow e=0$
	$m = 1,$	$x = [1, 0, 1] \rightarrow e=1$
	$m = 2,$	$x = [2, 0, 2] \rightarrow e=2$
<u>intervallum</u> eleje:	$m = 2,$	$x = [0] \rightarrow e=0$
<u>intervallum</u> vége:	$m = 2,$	$x = [2] \rightarrow e=2$
<u>több maximum:</u>	$m = 1,$	$x = [1, 0] \rightarrow e=1$

Számlálás tesztje: (1 .. n)

<u>intervallum</u> hossza:	$x = [0],$	$m = 2, \rightarrow e=0$
	$x = [0, 0],$	$m = 2, \rightarrow e=0$
	$x = [0, 0, 0],$	$m = 2, \rightarrow e=0$
<u>intervallum</u> eleje:	$x = [0, 1],$	$m = 2, \rightarrow e=0$
<u>intervallum</u> vége:	$x = [0, 1, 1],$	$m = 2, \rightarrow e=0$

Procedurális modul szerkezet



```
#include <vector>
#include <iostream>
#include <fstream>
using namespace std;

unsigned int mostFrequented(const vector<int> &x);
unsigned int frequency(const vector<int> &x, unsigned int i);

int main()
{ ... }

unsigned int mostFrequented(const vector<int> &x)
{ ... }
unsigned int frequency(const vector<int> &x, unsigned int i)
{ ... }
```

Fő program

```
int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }
    int m;
    cout << "A sorozat legnagyobb természetes száma: " ; cin >> m;

    vector<int> x;
    int e;
    while(f >> e) if(e>=0 && e<=m) x.push_back(e);

    if(x.size()==0) {
        cout << "Üres tömbben nincs értelme a kérdésnek.\n";
    } else {
        cout << "A megadott tömb leggyakoribb eleme: "
              << x[mostFrequented(x)];
    }
    cout << endl;

    return 0;
}
```

Fő program

```
unsigned int mostFrequented(const vector<int> &x)
{
    unsigned int max = frequency(x, 0);
    int e = 0;
    for(int i=0; i<m; ++i){
        unsigned int c = frequency(x, i+1);
        if(max < c){
            max = c; e = i+1;
        }
    }
    return e;
}

unsigned int frequency(const vector<int> &x, int i)
{
    unsigned int c = 0;
    for(unsigned int j=i+1; j<x.size(); ++j){
        if(x[j]==i) ++c;
    }
    return c;
}
```


Objektum orientált programterv

$$A = (x:\mathbb{Z}^n, m:\mathbb{Z}, e:\mathbb{Z})$$

$$Ef = (x = x' \wedge n > 0 \wedge m = m' \wedge \forall i \in [1..n]: x[i] \in [0..m])$$

$$Uf = (x = x' \wedge h = \bigcup_{i=1}^n \{x[i]\} \wedge e = \text{maxElem}(h))$$

$h : \text{Zsák}$

Összegzés (zsákolás),
majd egy értékadás

$h := \emptyset$	
$i = 1 .. n$	
	$h := h \cup \{x[i]\}$
$e := \text{maxElem}(h)$	

$h:$

		4	-5	-5	
4					
	4	8	4	8	4
		11	11	11	

Megoldások

Procedurális elvű megoldás

1. Ha $m < n/2$, akkor érdemes a $0..m$ intervallum azon elemét keresni, amelyik legtöbbször (maximum kiválasztás) fordul elő (számlálás) a sorozatban. Az összehasonlítások száma : $m + (m+1) \cdot n$

A maximum kiválasztás m összehasonlítást végez.
Minden $0..m$ közé eső számra a számlálás az n elemű sorozatot vizsgálja végig. Ez $m < n/2$ esetén jobb, mint a korábbi: $(n-1) + n \cdot (n-1)/2$

Típus központú megoldás

2. A zsákba levő elemek multiplicitásait egy $m+1$ hosszú tömbben tároljuk úgy, hogy az n elemű sorozat elemeinek bezsákolása (összegzés) ne igényeljen összehasonlítást. Ha külön tároljuk a leggyakoribb elemet, akkor ennek lekérdezése nem igényel összehasonlítást, de a karbantartásához egy összehasonlítás kell .
Az összehasonlítások száma : n

A zsákba dobás 1 összehasonlítás és 2 értékadás.
Az üres zsák létrehozása $m+2$ értékadás.

Fekete doboz tesztelés első része

Főprogram (összegzés-zsákolás) *tesztje* (1 .. n)

intervallum hossza: x=[2] → e=2

$$x = [3, -5] \rightarrow e = 3$$
$$x = [3, -2, 3, 5] \rightarrow e = 3$$

intervallum eleje: $x = [1, 1, 2, 2, 3]$ $\rightarrow e=1$

intervallum vége: $x = [3, 2, 2, 1, 1] \rightarrow e=2$

terhelés : $\mathbf{x} = [3, 2, \dots, 2, 1] \rightarrow e=2$

Fő program

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

```
using namespace std;
```

```
int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }
    int m;
    cout << "A sorozat legnagyobb természetes száma: " ; cin >> m;

    Bag h(m);
    int e;
    while(f >> e){
        if(e>=0 && e<=m) h.putIn(e);
    }

    cout << "A megadott tömb leggyakoribb eleme: "
        << h.maxElem() << endl;

    return 0;
}
```

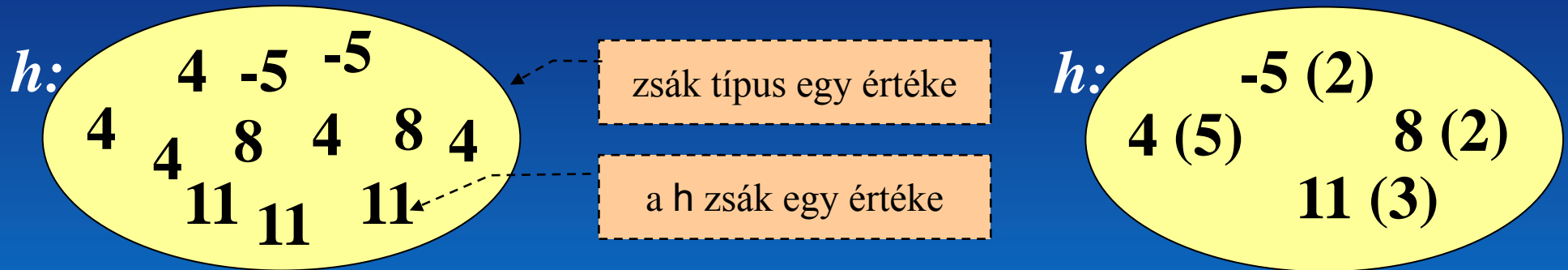
üres zsákot hoz létre

betesz a zsákba egy elemet

megadja a zsák leggyakoribb elemét

main.cpp

Zsák típus értékei



A zsák típus értékei (zsák típusú adatok, azaz zsákok) olyan multiplicitásos halmazok, amelyeket érték-darabszám párok halmazaként lehet megadni.

A zsák típusérték-halmaza ezeket a multiplicitásos halmazokat tartalmazza, azaz $2^{\mathbb{Z} \times \mathbb{N}}$ vagy $2^{\text{rec}(v:\mathbb{Z}, c:\mathbb{N})}$.

Zsák típus műveletei

létrehoz egy üres zsákot

$h := \emptyset$

$h: 2^{\mathbb{Z} \times \mathbb{N}}$

Betesz egy elemet a zsákba

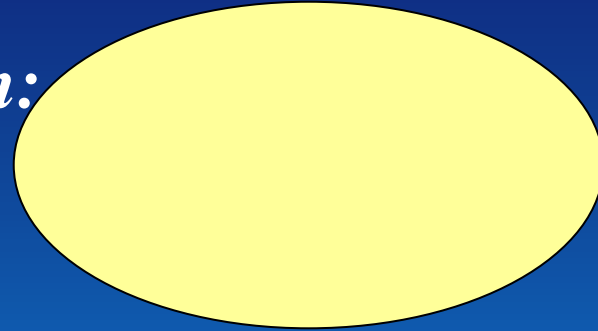
$h := h \cup \{e\}$ $e: \mathbb{Z}$

A zsák leggyakoribb eleme

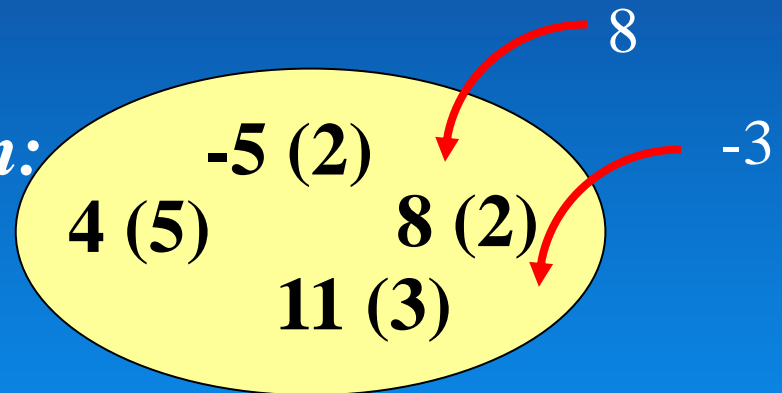
$e := \text{maxElem}(h)$

kivételt dob,
ha a zsák üres

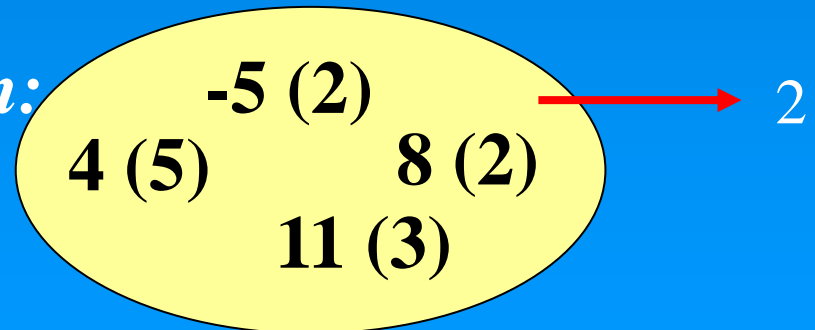
$h:$



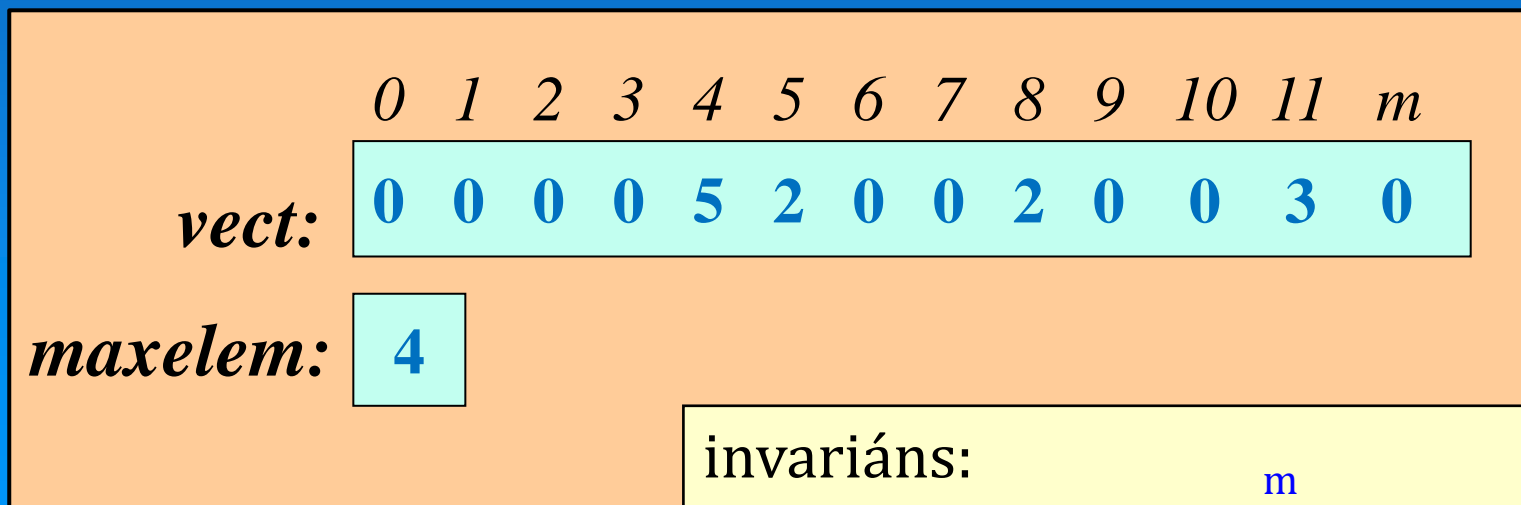
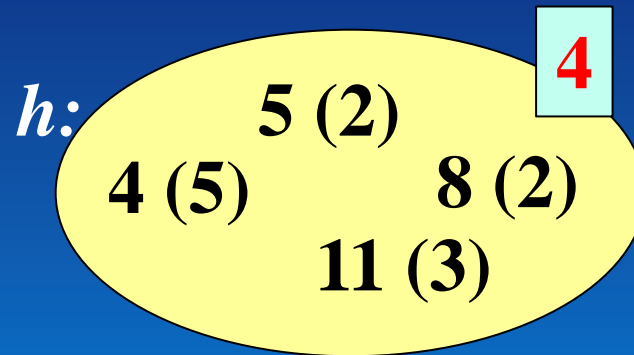
$h:$



$h:$



Zsák típus reprezentációja



invariáns:

$$\text{vect}[\text{maxelem}] = \max_{i=0}^m \text{vect}[i]$$

Zsák típus implementációja

$h := h \cup \{e\}$

$++vect[e]$

$vect[e] > vect[maxelem]$

$maxelem := e$

—

$h := \emptyset$

$i = 0 .. m$

$vect[i] := 0$

$maxelem := 0$

invariáns beállítása

invariáns helyreállítása

$e := \text{maxElem}(h)$

$vect[maxelem] > 0$

$e := maxelem$

—

kivételt dob,
ha a zsák üres

Zsák típus

Típus-specifikáció

típusértékek

$2^{\mathbb{Z} \times \mathbb{N}}$

létrehoz egy üres zsákot

$h := \emptyset \quad h : 2^{\mathbb{Z} \times \mathbb{N}}, m : \mathbb{N}$

betesz egy elemet a zsákba

$h := h \cup \{e\} \quad e : \mathbb{Z}$

zsák leggyakoribb eleme

$e := \text{max_elem}(h)$

műveletek

reprezentáció

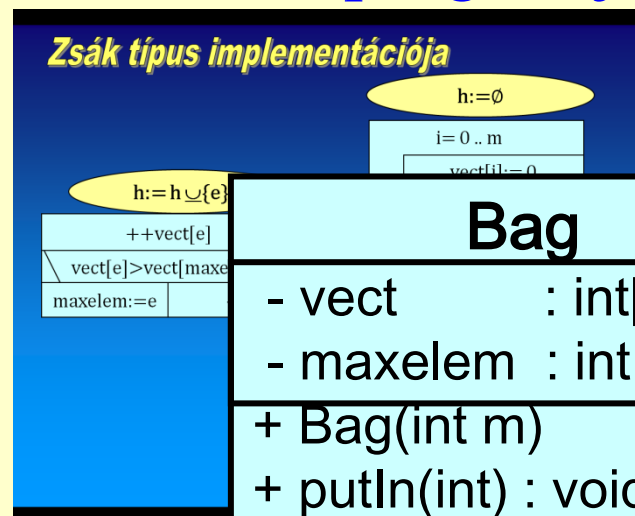
$\text{vect} : \mathbb{N}^{0..m}$

$\text{maxelem} : \mathbb{N}$

invariáns:

$\text{vect}[\text{maxelem}] = \max_{i=0}^m \text{vect}[i]$

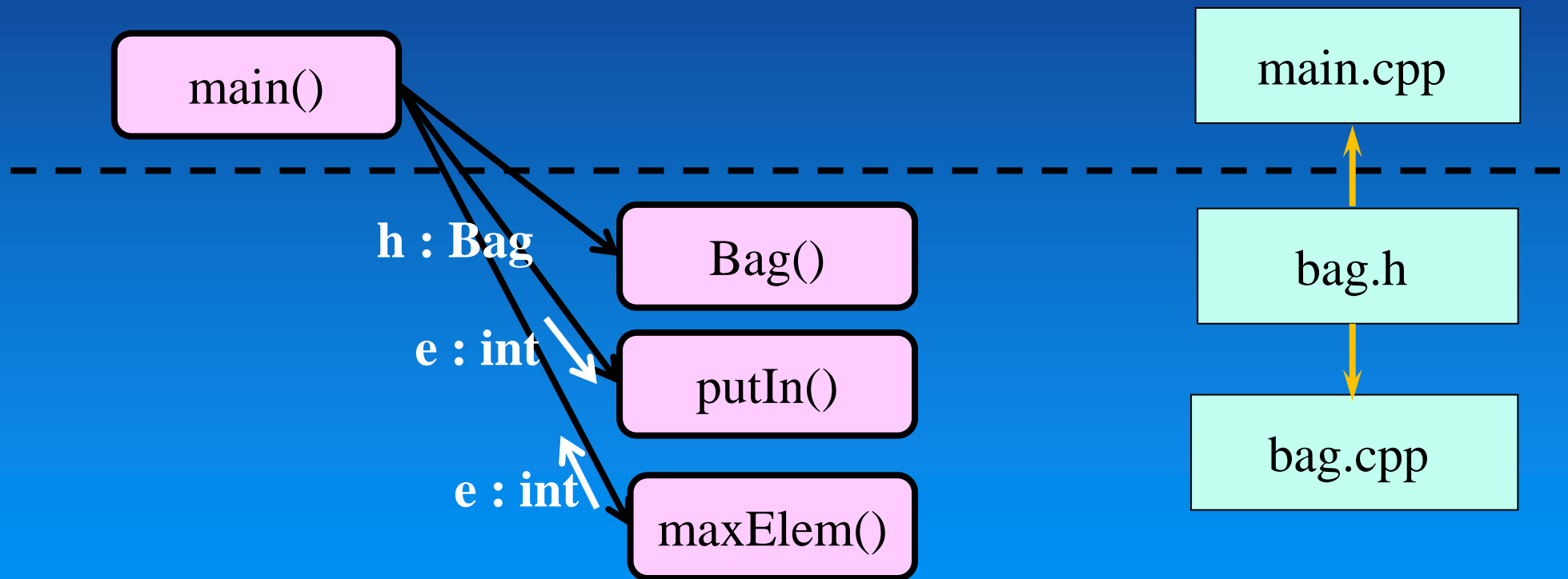
műveletek programjai



implementáció

Típus-megvalósítás

Modul szerkezet



Zsák típus osztálya

```
class Bag{
public:
    Bag(int m);

    void putIn(int e) {
        if(++_vect[e] > _vect[_maxelem]) _maxelem = e;
    }

    int maxElem() const;
private:
    std::vector<int> _vect;
    int _maxelem;
};
```

publikus metódusok

konstruktor, ami üres zsákot hoz létre
a `Bag h(23)` hatására

hívási mód: `h.putIn(5);`
a `h` zsák a kitüntetett paraméter, amelynek
`_vect` és `_maxind` adattagjaival dolgozik

hívási mód: `int a = h.maxElem();`
ez egy konstans metódus: nem változtatja
meg az adattagokat

privát adattagok

Zsák típus osztálya

```
#ifndef BAG_H
#define BAG_H
#include <vector>
```

```
class Bag{
public:
    enum Exceptions{EmptyBag};

    Bag(int m) {
        _vect.resize(m+1); _maxelem = 0;
        for(int i=0; i<m; ++i) _vect[i] = 0;
    }
    void putIn(int e) {
        if(++_vect[e] > _vect[_maxelem]) _maxelem = e;
    }
    int maxElem() const {
        if(_vect[_maxelem]>0) return _maxelem;
        else throw EmptyBag;
    }

private:
    std::vector<int> _vect;
    int _maxelem;
};
```

hibaesetek

kivétel dobás

```
#endif // BAG_H
```

```
#include <iostream>
#include <fstream>
#include "bag.h"
using namespace std;
int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }
    int m;
    cout << "A sorozat legnagyobb természetes száma: " ; cin >> m;
    Bag h(m);
    int e;
    while(f >> e){
        if(e>=0 && e<=m) h.putIn(e);
    }
    try{
        cout << "A megadott tömb leggyakoribb eleme: "
             << h.maxElem() << endl;
    }catch(Bag::Exceptions ex){
        if(ex==Bag::EmptyBag) { cout << "Üres a zsák!\n"; }
    }
    return 0;
}
```

```
#include <iostream>
#include <fstream>
#include "bag.h"
using namespace std;
int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }
    int m;
    cout << "A sorozat legnagyobb természetes száma: " ; cin >> m;
    Bag h(m);
    int e;
    while(f >> e){
        if(e>=0 && e<=m) h.putIn(e);
    }
    try{
        cout << "A megadott tömb leggyakoribb eleme: "
             << h.maxElem() << endl;
    }catch(Bag::Exceptions ex){
        if(ex==Bag::EmptyBag) { cout << "Üres a zsák!\n"; }
    }
    return 0;
}
```

kivétel figyelése

kivétel elkapása és kezelése

Fekete doboz tesztelés másik része

Zsák osztály tesztje

Bag() konstruktor tesztje:

- üres zsák jön-e létre

putIn() metódus tesztje:

- egy a zsákban még nem szereplő elem illetve egy már ott levő elem újbóli betevése

maxElem() metódus tesztje: (mintha maximum kiválasztás lenne)

- üres zsák esetén
- egy elemű zsák esetén
- több elemű zsák esetén, amelyben a leggyakoribb elem egyértelmű
- több elemű zsák, amelyben a leggyakoribb elem nem egyértelmű
- több elemű zsák, amelyben a legelőször betett elem a leggyakoribb
- több elemű zsák, amelyben a legutoljára betett elem a leggyakoribb

Integrációs teszt: (itt nem kritikus)

- $h := \emptyset$ utána sokszor lefut a $h := h \cup \{e\}$
- ...

Minden nem konstans metódusnál ellenőrizni kell az invariáns meglétét

1. változat

Melyik a leggyakoribb elem egy egész számokból álló sorozatban?

Procedurális programterv

$$A = (x:\mathbb{Z}^n, e:\mathbb{Z})$$

$$Ef = (x = x' \wedge n > 0)$$

maximum kiválasztásban
számlálás, majd egy értékadás

$$Uf = (x = x' \wedge (\max, \text{ind}) = \mathbf{MAX}_{i=1}^n \text{hány}(i) \wedge e = x[\text{ind}])$$

ahol $\text{hány}(i) = \sum_{j=1}^{i-1} 1$

$$x[j] = x[i]$$

$$s := \text{hány}(i)$$

max, ind := 0, 1

i = 2 .. n

s := hány(i)

max < s

max, ind := s, i

—

e := x[ind]

s := 0

j = 1 .. i-1

x[j] = x[i]

s := s + 1

—

Fekete doboz tesztelés vázlata

Maximum kiválasztás tesztje: (1 .. n)

<u>intervallum</u> hossza:	$x = [2]$	$\rightarrow e=2$
	$x = [3, -5]$	$\rightarrow e=3$
	$x = [3, -2, 3, 5]$	$\rightarrow e=3$
<u>intervallum</u> eleje:	$x = [1, 1, 2, 2, 3]$	$\rightarrow e=1$
<u>intervallum</u> vége:	$x = [3, 2, 2, 1, 1]$	$\rightarrow e=1$
<u>több maximum</u> :	$x = [1, 2, 2, 3, -4, -4, 7]$	$\rightarrow e=2$
<u>érvénytelen</u> input:	$ x = 0$	

Számlálás tesztje: (i+1 .. n)

a tesztadatok megegyeznek a fentiekkel

<u>intervallum</u> hossza:	$x = [2]$	$\rightarrow e=2$
	$x = [3, -5]$	$\rightarrow e=3$
	$x = [3, -2, 3, 5]$	$\rightarrow e=3$
<u>intervallum</u> eleje:	$x = [1, 1, 2, 2, 3]$	$\rightarrow e=2$
<u>intervallum</u> vége:	$x = [3, 2, 2, 1, 1]$	$\rightarrow e=2$

Procedurális modul szerkezet



```
#include <vector>
#include <iostream>
#include <fstream>
using namespace std;

unsigned int mostFrequented(const vector<int> &x);
unsigned int frequency(const vector<int> &x, unsigned int i);

int main()
{ ... }

unsigned int mostFrequented(const vector<int> &x)
{ ... }
unsigned int frequency(const vector<int> &x, unsigned int i)
{ ... }
```

Fő program

```
int main()
{
    ifstream f("input.txt");
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }

    vector<int> x;
    int e;
    while(f >> e) x.push_back(e);

    if(x.size()==0) {
        cout << "Üres tömbben nincs értelme a kérdésnek.\n";
    } else {
        cout << "A megadott tömb leggyakoribb eleme: "
              << x[mostFrequented(x)];
    }
    cout << endl;

    return 0;
}
```

Fő program

```
unsigned int mostFrequented(const vector<int> &x)
{
    unsigned int max = 0;
    unsigned int ind = 0;
    for(unsigned int i=0; i<x.size(); ++i){
        int c = frequency(x, i);
        if(max < c){
            max = c; ind = i;
        }
    }
    return ind;
}

unsigned int frequency(const vector<int> &x, unsigned int i)
{
    unsigned int c = 0;
    for(unsigned int j=0; j<i; ++j){
        if(x[j]==x[i]) ++c;
    }
    return c;
}
```

Objektum orientált programterv

$$A = (x:\mathbb{Z}^n, e:\mathbb{Z})$$

$$Ef = (x = x' \wedge n > 0)$$

$$Uf = (x = x' \wedge h = \bigcup_{i=1}^n \{x[i]\} \wedge e = \text{maxElem}(h))$$

$h : \text{Zsák}$

Összegzés (zsákolás),
majd egy értékadás

$h := \emptyset$

$i = 1 \dots n$

$h := h \cup \{x[i]\}$

$e := \text{maxElem}(h)$

$h:$

	4	-5	-5		
4		8	4	8	4
	4				
		11	11	11	

Megoldások hatékonysága

Procedurális elvű megoldás

1. Maximum kiválasztás az n elemű sorozat elemeinek előfordulási számai (számlálás) felett.

összehasonlítások száma : $(n-1) + n \cdot (n-1)/2$

A maximum kiválasztás $n-1$ összehasonlítást végez. Ezen felül minden elemre megszámlaljuk, hogy hányszor fordul elő a tömb hátralevő részében: az i -dik elemnél ez $n-i$ összehasonlítás.

Típus központú megoldás

2. Az n elemű sorozat elemeinek bezsákolása (összegzés), ha egy elem zsákba dobásához egy lineáris keresést használunk.
(A zsák leggyakoribb elemének kivétele konstans futási idejűvé tehető.)

összehasonlítások száma : $\sim n \cdot (n-1)/4 + 2 \cdot n$

A lineáris keresés átlagosan a zsákba korábban betett különböző elemeknek a felével végez összehasonlítást, amit még további két feltétel-kiértékelés követ. Mindez n -szer történik meg.

Fekete doboz tesztelés első része

Főprogram (összegzés-zsákolás) *tesztje* (1 .. n)

intervallum hossza: x=[2] → e=2

$$x = [3, -5] \rightarrow e = 3$$
$$x = [3, -2, 3, 5] \rightarrow e = 3$$

intervallum eleje: $x = [1, 1, 2, 2, 3]$ $\rightarrow e=1$

intervallum vége: $x = [3, 2, 2, 1, 1] \rightarrow e=2$

terhelés : $\mathbf{x} = [3, 2, \dots, 2, 1] \rightarrow e=2$

Fő program

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

```
int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }

    Bag h;
    int e;
    while(f >> e){
        h.putIn(e);

    }

    try{
        cout << "A megadott tömb leggyakoribb eleme: "
              << h.maxElem() << endl;
    } catch (Bag::Exceptions ex){
        if(ex==Bag::EmptyBag){ cout << "Üres a zsák!\n"; }
    }
    return 0;
}
```

üres zsákot hoz létre

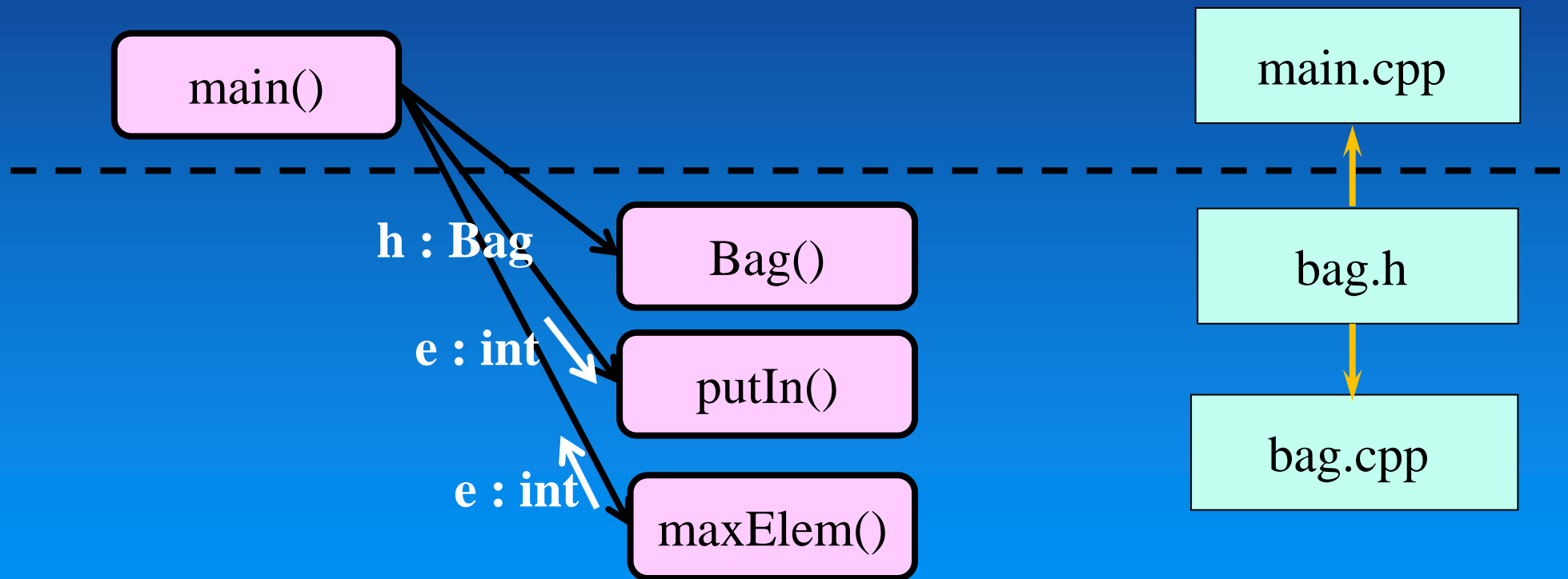
betesz a zsákba egy elemet

megadja a zsák leggyakoribb elemét

zsák műveletek kivétele

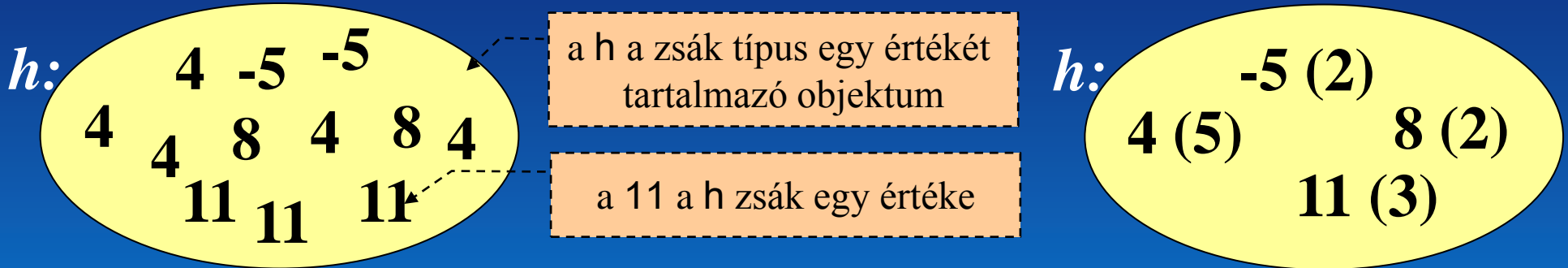
main.cpp

Modul szerkezet



Zsák típus értékei

Ugyanaz a zsák kétféle formában



A zsák típus értékei (zsák típusú adatok, azaz zsákok) olyan multiplicitásos halmazok, amelyeket érték-darabszám párok halmazaként lehet megadni.

A zsák típusérték-halmaza ezeket a multiplicitásos halmazokat tartalmazza, azaz $2^{\mathbb{Z} \times \mathbb{N}}$ vagy $2^{\text{rec}(v:\mathbb{Z}, c:\mathbb{N})}$.

Zsák típus műveletei

létrehoz egy üres zsákot

$h := \emptyset$

$h: 2^{\mathbb{Z} \times \mathbb{N}}$

Betesz egy elemet a zsákba

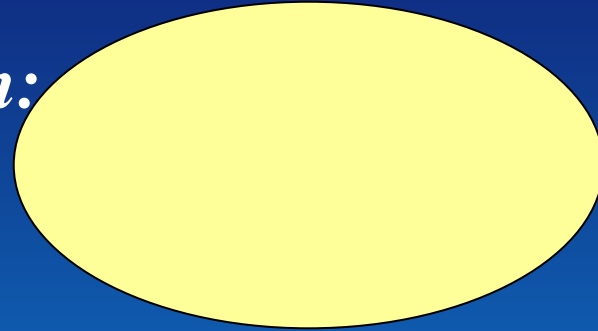
$h := h \cup \{e\} \quad e: \mathbb{Z}$

A zsák leggyakoribb eleme

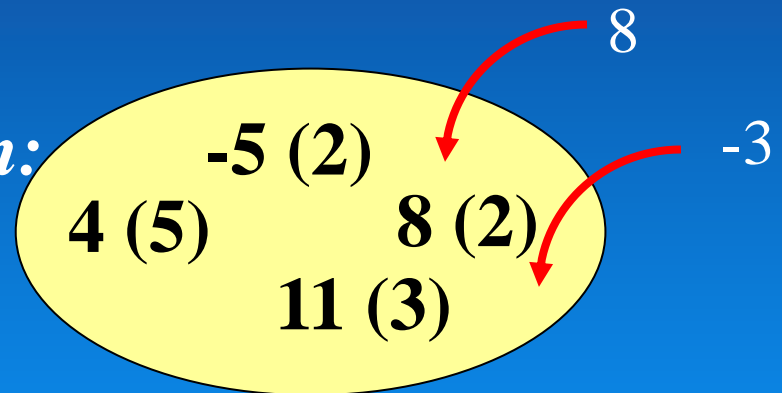
$e := \text{maxElem}(h)$

kivételt dob,
ha a zsák üres

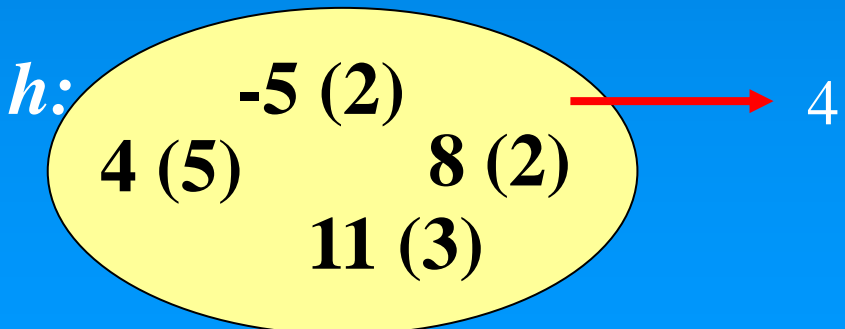
$h:$



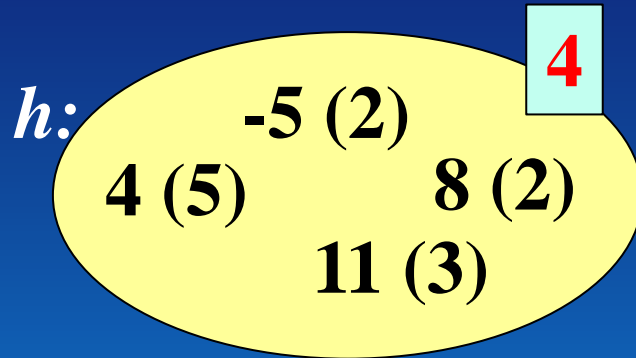
$h:$



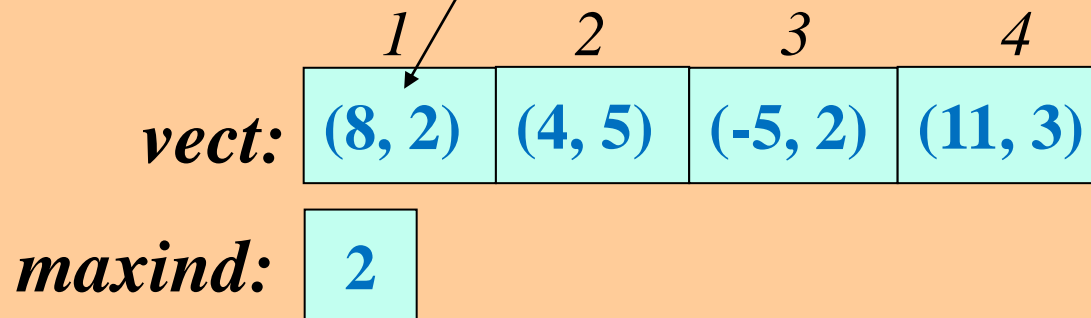
$h:$



Zsák típus reprezentációja



$\text{rec}(v:\mathbb{Z}, c:\mathbb{N})$



invariáns:

$| \text{vect} | > 0 \rightarrow \text{vect}[\text{maxind}].c = \max_{i=1}^{|\text{vect}|} \text{vect}[i].c$

Zsák típus implementációja

$h := \emptyset$

$\text{vect} := \langle \rangle$

$h := h \cup \{e\}$

$l, \text{ind} = \text{search}_{j=1 \dots |\text{vect}|} (\text{vect}[j].v = e)$

1

$++\text{vect}[\text{ind}].c$

$\text{vect} := \text{vect} \oplus (e, 1)$

$\text{vect}[\text{maxind}].c < \text{vect}[\text{ind}].c$

$|\text{vect}| = 1$

$\text{maxind} := \text{ind}$

–

$\text{maxind} := 1$

–

invariáns helyreállítása

kivételt dob,
ha a zsák üres

$e := \text{maxElem}(h)$

$|\text{vect}| \neq 0$

$e := \text{vect}[\text{maxind}].v$

–

Zsák típus

Típus-specifikáció

típusértékek

$$2^{\mathbb{Z} \times \mathbb{N}}$$

létrehoz egy üres zsákot

$$h := \emptyset \quad h : 2^{\mathbb{Z} \times \mathbb{N}}$$

betesz egy elemet a zsákba

$$h := h \cup \{e\} \quad e : \mathbb{Z}$$

zsák leggyakoribb eleme

$$e := \text{max_elem}(h)$$

műveletek

reprezentáció

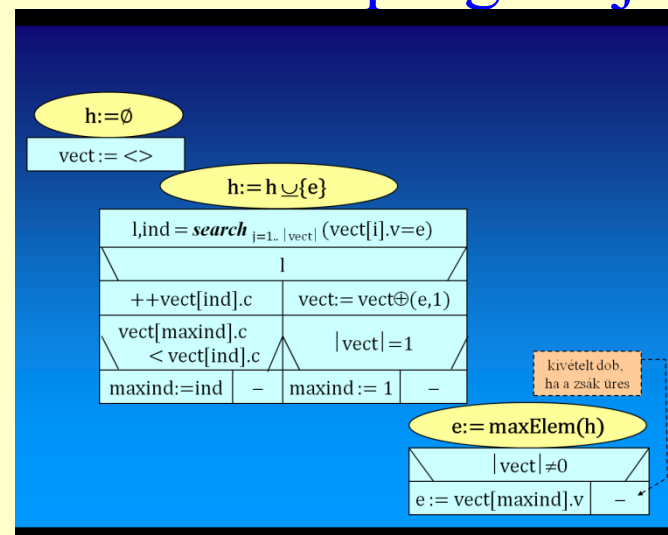
$$\text{vect} : (\text{rec}(v:\mathbb{Z}, c:\mathbb{N}))^*$$

$$\text{maxind} : \mathbb{N}$$

invariáns:

$$|\text{vect}| > 0 \rightarrow \text{vect}[\text{maxind}].c = \max_{i=1}^{|\text{vect}|} \text{vect}[i].c$$

műveletek programjai



implementáció

Típus-megvalósítás

Zsák típus osztálya

```
#ifndef BAG_H
#define BAG_H
#include <vector>
```

```
class Bag{
public:
    Bag() { _vect.clear(); }
    void putIn(int e);
    int maxElem() const;
private:
    std::vector<Pair> _vect;
    unsigned int _maxind;

    bool search(int e, unsigned int &ind) const;
};
```

```
#endif // BAG_H
```

inline definíció

konstans metódus

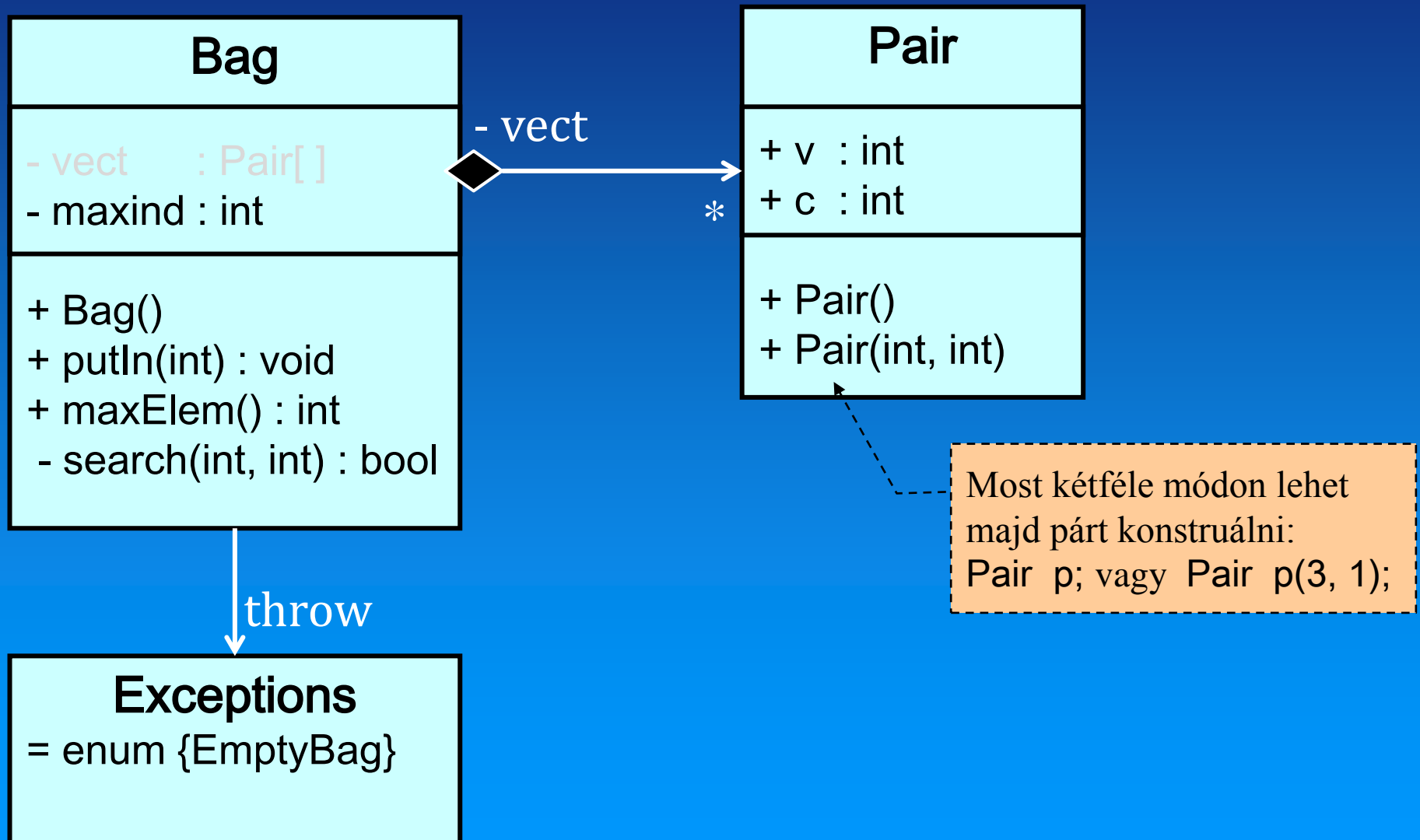
Bag

- vect : Pair[]
- maxind : int

+ Bag()
+ putIn(int) : void
+ maxElem() : int
- search(int, int) : bool

privát metódus

Zsák típus osztály diagramja



Zsák típus osztálya

```
#ifndef BAG_H
#define BAG_H
#include <vector>
```

```
class Bag{
public:
    enum Exceptions{EmptyBag};

    Bag() { _vect.clear(); }
    void putIn(int e);
    int maxElem() const;
private:
    struct Pair{
        int v;
        int c;
        Pair(){}
        Pair(int a, int b): v(a), c(b) {}
    };
    std::vector<Pair> _vect;
    unsigned int _maxind;

    bool search(int e, unsigned int &ind) const;
};
```

az **enum** is egy típust definiál,
akárcsak mint a **class**

a **struct** olyan, mint a **class**, csak
más láthatósági szabályokkal

Ha csak az üres konstruktor kellene,
akkor ezeket nem kellene megadni.

Itt a **Pair** első konstruktorát használjuk.

```
#endif // BAG_H
```

Zsák típus osztálya

```
#include "bag.h"
```

```
using namespace std;
```

```
void Bag::putIn(int e)
```

a Bag osztályhoz tartozó

```
{
```

```
    unsigned int ind;
```

```
    if (search(e, ind)) {
```

```
        ++_vect[ind].c;
```

```
        if (_vect[_maxind].c < _vect[ind].c) _maxind = ind;
```

```
    } else {
```

```
        _vect.push_back(Pair(e, 1));
```

Itt használjuk a Pair
második konstruktorát.

```
        if (_vect.size() == 1) _maxind = 0;
```

```
    }
```

```
}
```

```
int Bag::maxElem() const
```

```
{
```

```
    if (_vect.size() == 0) throw EmptyBag;
```

```
    return _vect[_maxind].v;
```

```
}
```

```
bool Bag::search(int e, unsigned int &ind) const
```

```
{
```

```
    bool l = false;
```

```
    for(unsigned int i=0; !l && i<_vect.size(); ++i){
```

```
        l = e == _vect[i].v; ind = i;
```

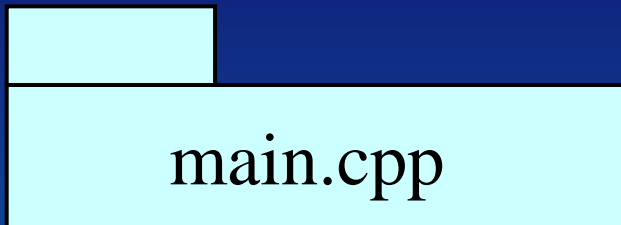
```
    }
```

```
    return l;
```

```
};
```

bag.cpp

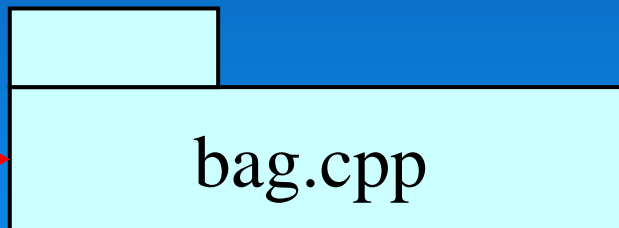
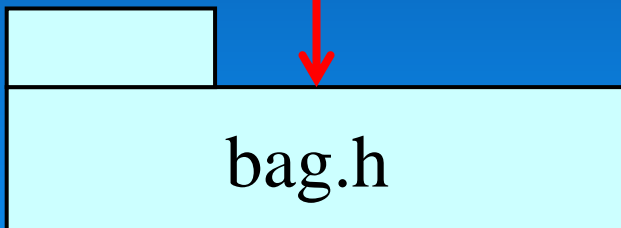
Csomag szerkezet



```
#include <iostream>
#include <vector>
#include "bag.h"

using namespace std;

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



```
#ifndef _BAG_H
#define _BAG_H

class Bag{...};

#endif // _BAG_H
```

```
#include "bag.h"

void Bag::putIn(int e) {...}
int Bag::maxElem() const {...}
bool Bag::search(...) const {...}
```

Fekete doboz tesztelés másik része

Zsák osztály tesztje

Bag() konstruktor tesztje:

- üres zsák jön-e létre

Minden nem konstans metódusnál ellenőrizni kell az invariáns meglétét

putIn() metódus (azon belül a *search()*) tesztje:

- első elem betevése, másodikként betett elem megegyezik az elsővel illetve eltér tőle, a harmadikként betett elem az elsővel illetve a másodikkal azonos
- intervallum hossza: a zsákban levő különböző elemek eltérő száma szerint
- intervallum eleje: több elemű zsák, amelybe a legelőször betett elemet tesszük be újra
- intervallum vége: több elemű zsák, amelyben az utoljára betett elemet tesszük be újra
- funkció: a betenni kívánt elem még nincs, illetve már benn van a zsákban

maxElem() metódus tesztje:

- üres zsák esetén
- egy elemű zsák esetén
- több elemű zsák esetén, amelyben a leggyakoribb elem egyértelmű
- több elemű zsák, amelyben a leggyakoribb elem nem egyértelmű
- több elemű zsák, amelyben a legelőször betett elem a leggyakoribb
- több elemű zsák, amelyben a legutoljára betett elem a leggyakoribb

Integrációs teszt: (itt nem kritikus)

- $h := \emptyset$ utána sokszor lefut a $h := h \cup \{e\}$
- ...

Másik objektum-orientált változat

```
#include "menu.h"
int main() {
    Menu m;
    m.run();
    return 0;
};
```

main()

create
run()

m : Menu

create
putIn()
maxElem()

h : Bag

1 : (putIn())
2 : (maxElem())

Menu

- h : Bag

+ Menu()
+ run() : void
- menuWrite() : void
- point1_putIn() : void
- point2_maxElem() : void

- h

Bag

- vect : Pair[]

- maxind : int

+ Bag()
+ putIn(int) : void
+ maxElem() : int
- search(int, int) : bool

- vect

Pair

+ v : int
+ c : int

+ Pair()
+ Pair(int, int)

*

Menu

```
#include "menu.h"
#include <iostream>
using namespace std;
bool onetwo(int n) { return n==1 || n==2; }
void Menu::run()
{
    int n = 0;
    do{
        menuWrite();
        n = read_int("válassz: ", "1 vagy 2", onetwo);
        switch (n) {
            case 1: point1_putIn(); break;
            case 2: point2_maxElem(); break;
        }
    } while (n!=0);
}
void Menu::menuWrite()
{
    cout << "0 - kilépés" << endl;
    cout << "1 - elem behelyezése" << endl;
    cout << "2 - leggyakoribb elem" << endl;
}
```

```
#include "bag.h"
```

```
class Menu{
public:
    void run();
private:
    Bag h;

    void menuWrite();
    void point1_putIn();
    void point2_maxElem();
};
```

menu.h

Menu

- h : Bag

+ Menu()
+ run() : void
- menuWrite() : void
- point1_putIn() : void
- point2_maxElem() : void

menu.cpp

Menu

```
void Menu:: point1_putIn()
{
    int e;
    cout << "Elem: ";
    cin >> e;
    h.putIn(e);
}

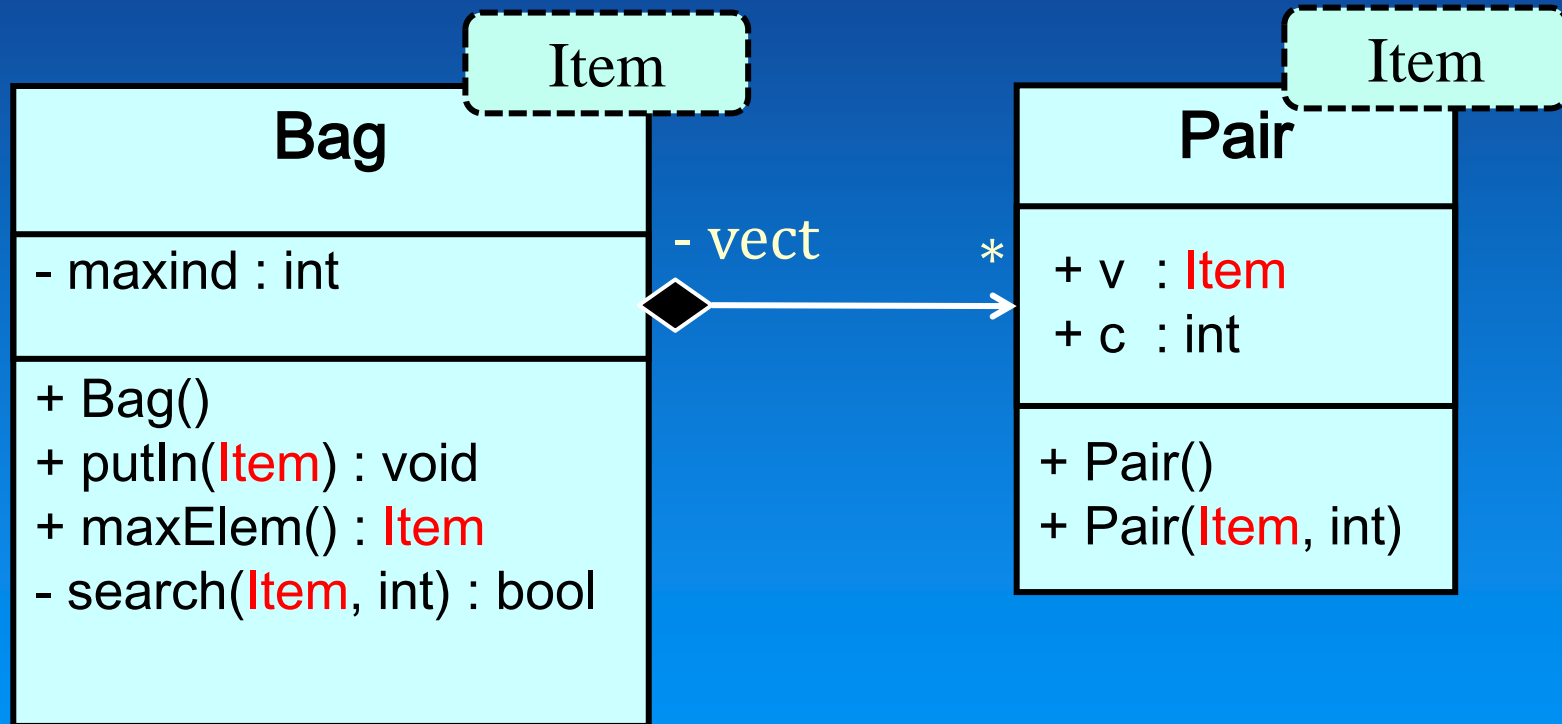
void Menu:: point2_maxElem()
{
    try{
        cout << "A leggyakoribb elem: "
              << h.maxElem() << endl;
    } catch (Bag::Exceptions ex) {
        if (ex==Bag::EmptyBag) { cout << "Üres a zsák!\n"; }
    }
    cout << endl;
}
```


2. változat

*Melyik a leggyakoribb elem egy **adott típusú elemekből** álló sorozatban ?*

Most csak az objektum orientált megoldást nézzük meg egy olyan zsákkal, amelynek megvalósítása a korábban bemutatott, tetszőleges egész számok befogadására képes zsákéra hasonlít.

Osztály diagram



Fő program

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

int main()
{
    ifstream f( "input.txt" );
    if(f.fail()){
        cout << "Hiba a fájl nyitásakor!\n";
        return 1;
    }

    Bag<int> h;
    int e;
    while(f >> e){
        h.putIn(e);
    }

    try{
        cout << "A megadott tömb leggyakoribb eleme: "
              << h.maxElem() << endl;
    }catch(Bag<int>::Exceptions ex){
        if(ex==Bag<int>::EmptyBag) { cout << "Üres a zsák!\n"; }
    }
    return 0;
}
```

main.cpp

Zsák típus osztálya

```
template <typename Item>
class Bag{
public:
    enum Exceptions{EmptyBag};

    Bag() { _vect.clear(); }
    void putIn(const Item &e);
    Item maxElem() const;
private:
    struct Pair{
        Item v;
        int c;
        Pair(){}
        Pair(const Item &a, int b): v(a), c(b) {}
    };
    std::vector<Pair> _vect;
    unsigned int _maxind;

    bool search(const Item &e, unsigned int &ind) const;
};
```

Zsák típus osztálya

```
template <typename Item>
void Bag<Item>::putIn(const Item &e)
{
    unsigned int ind;
    if (search(e, ind)) {
        ++_vect[ind].c; if(_vect[_maxind].c<_vect[ind].c) _maxind=ind;
    } else {
        _vect.push_back(Pair(e,1)); if(_vect.size()==1) _maxind=0;
    }
}

template <typename Item>
Item Bag<Item>::maxElem() const
{
    if (_vect.size()==0 ) throw EmptyBag;
    return _vect[_maxind].v;
}

template <typename Item>
bool Bag<Item>::search(const Item &e, unsigned int &ind) const
{
    bool l = false;
    for(unsigned int i=0; !l && i<_vect.size(); ++i){
        l = e == _vect[i].v; ind = i;
    }
    return l;
};
```

Csomag diagram

Egy sablon önmagában nem fordítható,
ezért minden elemét egy közös fejlőlmányba tesszük.

bag.h

```
class Bag{
    private:
        ...
    public:
        ...
        struct Pair{
            int v;
            ...
        };
};
```

bag.cpp

```
void Bag::putIn(int e)
{...}
int Bag::maxElem()
const {...}
...
```

bag.hpp

```
template <typename Item>
class Bag{
    private:
        ...
    public:
        ...
        struct Pair{
            Item v;
            ...
        };
};

template <typename Item>
void Bag<Item>::putIn(const Item &e)
{...}

template <typename Item>
Item Bag<Item>::maxElem() const
{...}
```

Kitérő: Beolvasás sablon

```
#include "read_template.hpp"
bool all_int(int) {return true;}
int main()
{
    int n = read<int>("Egész szám: ", "Hiba", all_int);
    return 0;
}
```

main.cpp

```
#ifndef READ_HPP
#define READ_HPP
#include <iostream>
#include <string>

template <typename Item>
Item read( const std::string &msg, const std::string &err,
          bool valid(Item))
{
    Item n;
    bool hiba;
    do{
        std::cout << msg;
        std::cin >> n;
        if((hiba=std::cin.fail())) std::cin.clear();
        std::string tmp=""; getline(std::cin, tmp);
        hiba = hiba || tmp.size()!=0 || !valid(n);
        if(hiba) std::cout << err << std::endl;
    }while(hiba);
    return n;
}

#endif // READ_HPP
```

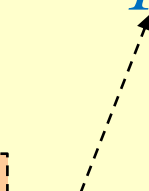
Az Item-re értelmezett kell
legyen a >> operator

read.hpp

3. változat

*Melyik a leggyakoribb elem egy **különböző típusú elemekből** álló sorozatban?*

Például egész számok
és sztringek vegyesen

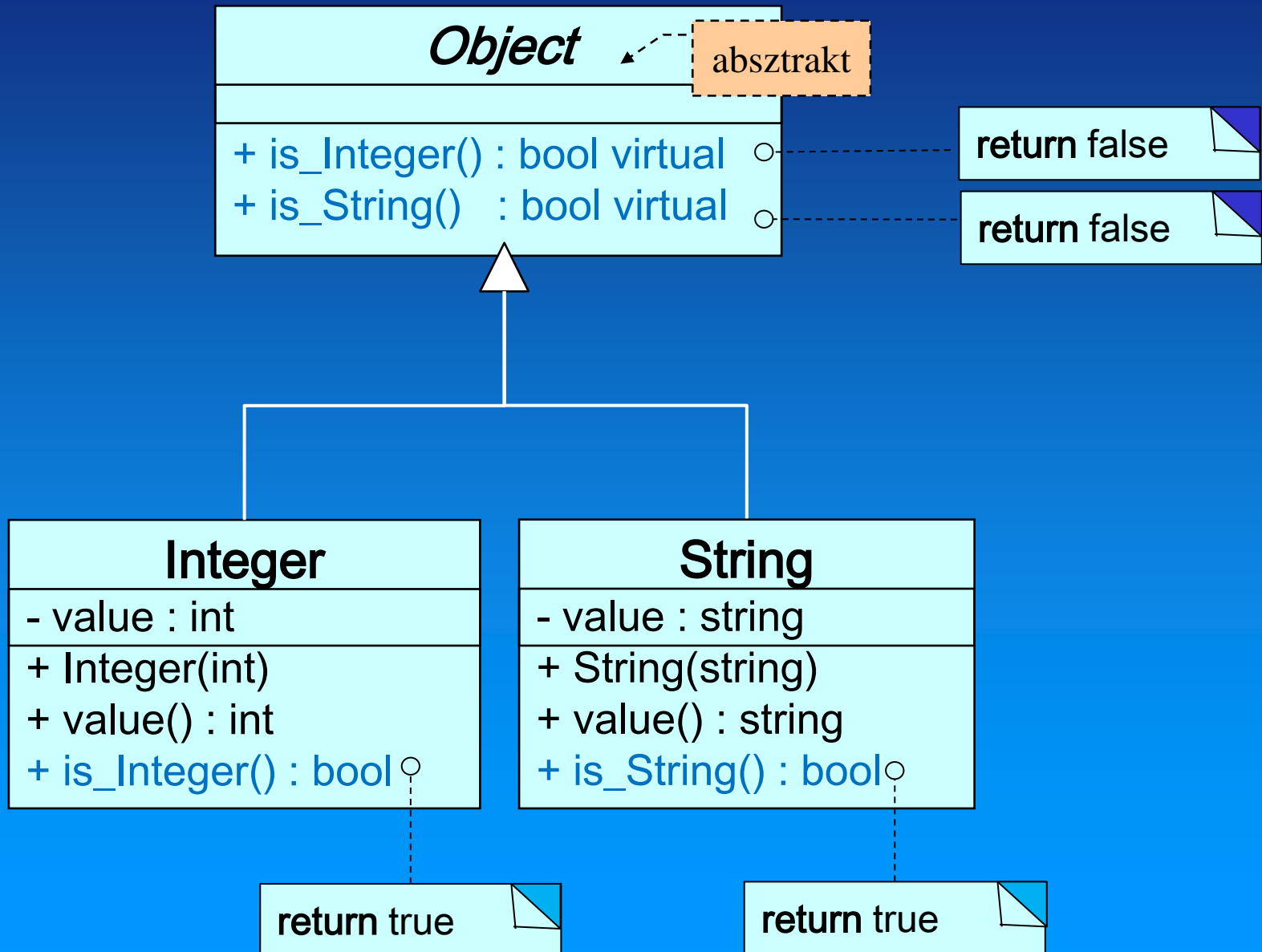


Most is csak az objektum orientált megoldást nézzük meg, és az előbb látott zsák-reprezentációt módosítjuk úgy, hogy egy zsákba eltérő típusú elemek is bekerülhessenek.

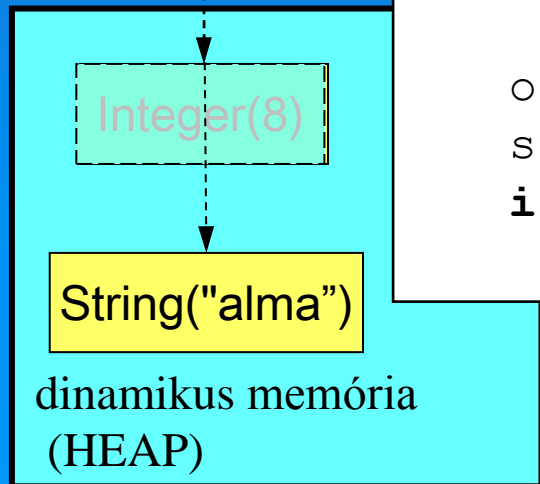
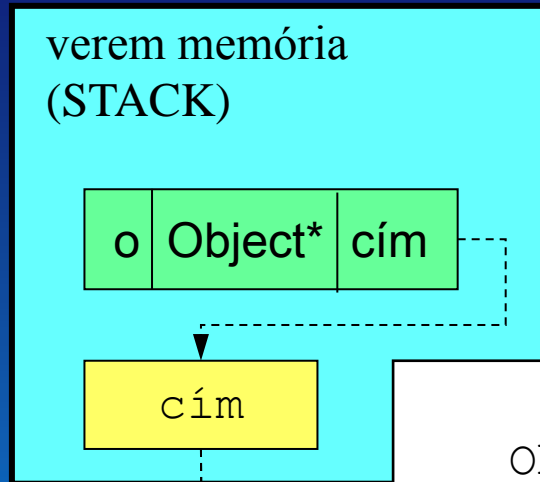
Alternatív szerkezetű típus

- ❑ Olyan típus kellene, amelynek **értékei különféle más típusokhoz tartozó értékek** lehetnek. Egy ilyen érték – alternatív módon – az alkotó típusok valamelyikéhez tartozik.
 - $T = \text{alt}(T_1, \dots, T_n)$
 - ha $t:T$ és $t_1:T_1, t_2:T_2$, akkor a $t := t_1$ és a $t := t_2$ is értelmes.
- ❑ Az alternatív szerkezetű típusnak egy értékéről (objektumáról, változójáról) fontos tudni, hogy az **melyik alkotó típusához tartozik**.
 - $T = \text{alt}(s_1: T_1, \dots, s_n: T_n)$ és $e \in T$ esetén a $e.s_i$ akkor igaz, ha $e \in T_i$.
 - Legyen $t:T, t_1:T_1, t_2:T_2$. A $t := t_1$ után a $t.s_1$ igaz, a $t.s_2$ hamis; $t := t_2$ után pedig a $t.s_2$ igaz és a $t.s_1$ hamis.

Alternatív szerkezetű típus megvalósítása



Alternatív szerkezetű objektum



Az `o` egy `Object` típusú pointer változó, amely egy `Object` vagy egy abból származtatott osztály objektumának memóriacímét tartalmazhatja. Az `o` egy címet, a `*o` pedig azon a címen található értéket (objektumot) jelöli.

A `new` helyet foglal egy `Integer` objektumnak, elhelyezi benne a `8` értékét, és visszaadja a helyfoglalás címét.

```
Object *o;
```

```
o = new Integer(8);
```

```
if( o->is_String() ) ...
```

```
if( o->is_Integer() )
```

`(*o).is_String()`

```
int a = ((Integer*)o)->value() + 4;
```

```
delete o;
```

memória felszabadítás:

```
o = new String("alma");
```

```
string a = ((String*)o)->value() + "körte";
```

```
if( o->is_String() ) ...
```

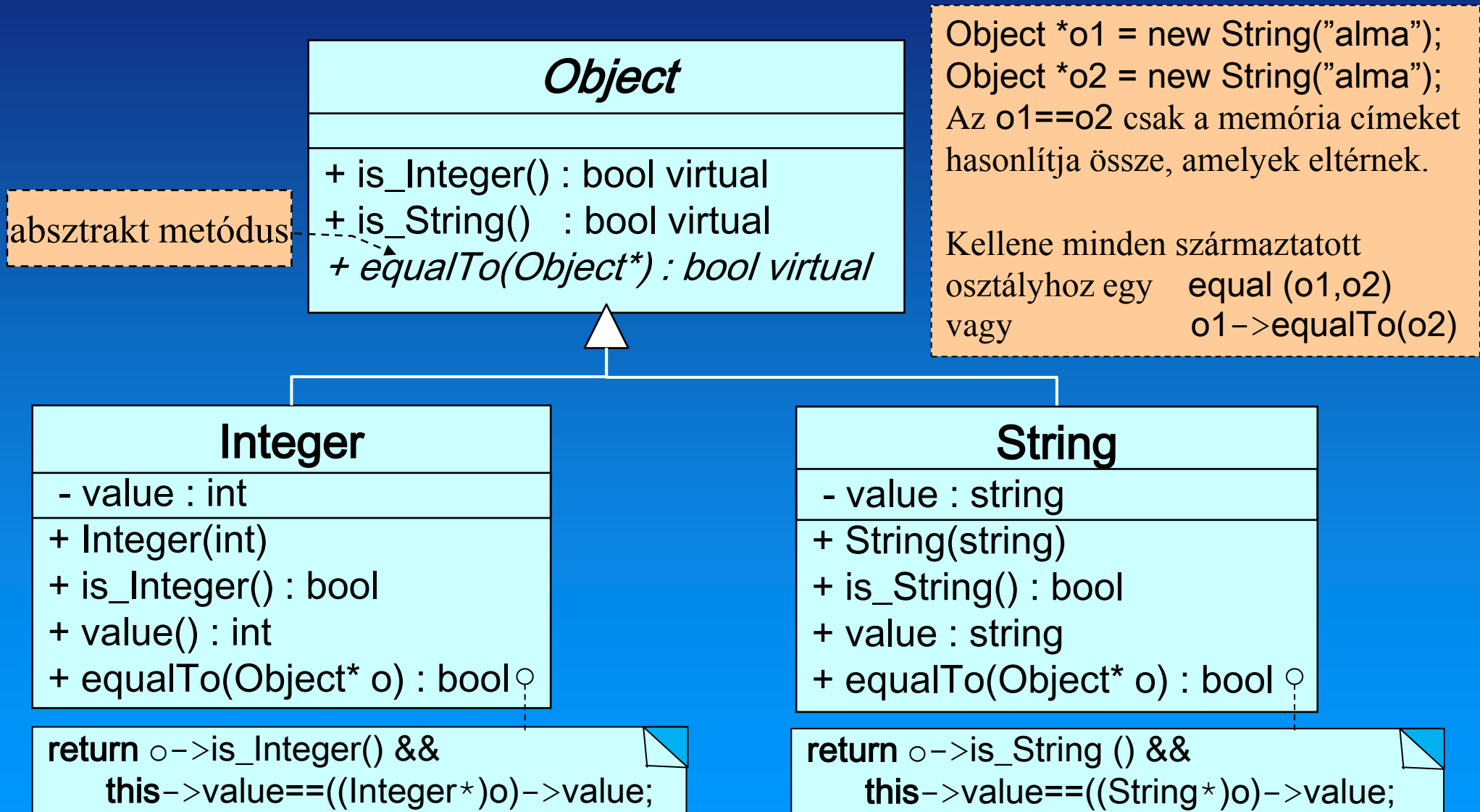
típus kényszerítés:

`o->value()` nem elég

Dinamikus kötés:

Annak az objektumnak az `is_String()` metódusa hajtódik végre, amely címére az `o` pointer éppen mutat. Ez futási időben dől csak el.

Alternatív szerkezetű objektumok összehasonlítása



Item osztály és leszármazottai

```
class Object {  
public:  
    virtual bool is_Integer() const {return false;}  
    virtual bool is_String() const {return false;}  
    virtual bool equalTo(const Object* r) const = 0;  
    virtual ~Object() {}  
};
```

felüldefiniálható

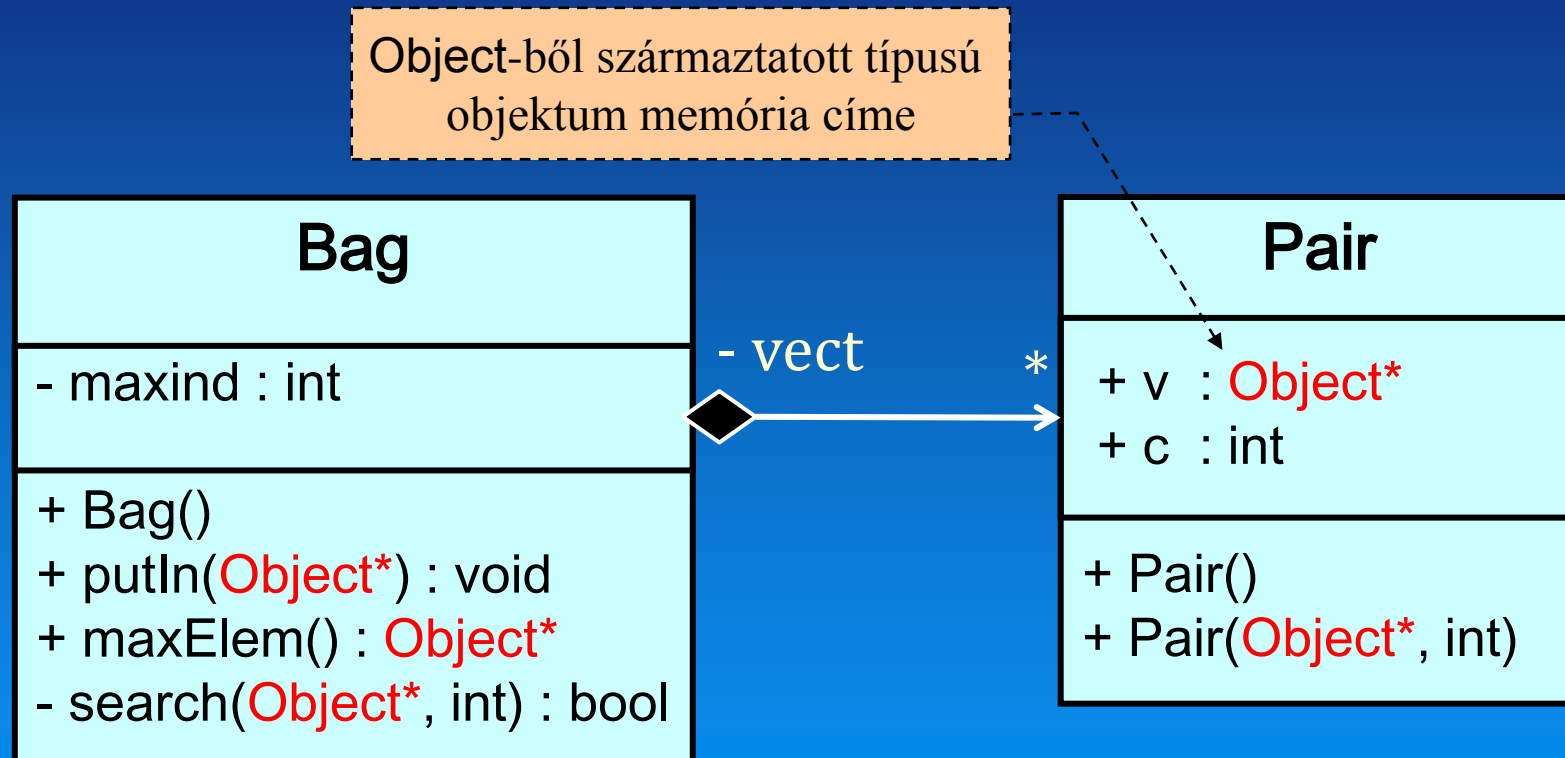
absztrakt metódus

```
class String : public Object  
private:  
    std::string _value;  
public:  
    String(const std::string str) : _value(str) {}  
    bool is_String() const { return true;}  
    std::string value() const { return _value; }  
    bool equalTo(const Object* o) const {  
        return o->is_String() &&  
        this->_value==( (String*)o)->_value;  
    }  
};
```

```
class Integer : public Object {  
private:  
    int _value;  
public:  
    Integer(int n) : _value(n) {}  
    bool is_Integer() const { return true;}  
    int value() const { return _value;}  
    bool equalTo(const Object* o) const {  
        return o->is_Integer() &&  
        this->_value==( (Integer*)o)->_value;  
    }  
};
```

object.h

Osztály diagram



Zsák típus osztálya

```
#include "object.h"
#include <vector>

class Bag{
public:
    enum Exceptions{EmptyBag};
    Bag() { _vect.clear(); }
    void putIn(Object* e);
    Object* maxElem() const;
private:
    struct Pair {
        Object* v;
        int c;
        Pair() {}
        Pair(Object* a, int b): v(a), c(b) {}
    };
    std::vector<Pair> _vect;
    unsigned int _maxind;

    bool search(const Object* e, unsigned int &ind) const;
};
```

Zsák típus osztálya

```
void Bag::putIn(Object* e)
{
    unsigned int ind;
    if (search(e, ind)){
        ++_vect[ind].c; if(_vect[_maxind].c<_vect[ind].c) _maxind = ind;
    }else {
        _vect.push_back(Pair(e,1)); if(_vect.size() == 1) _maxind = 0;
    }
}
```

```
Object* Bag::maxElem() const
{
    if (_vect.size()==0 ) throw EmptyBag;
    return _vect[_maxind].v;
}
```

Dinamikus kötés:

Az **e**-ben tárolt címen található,
Item-ből származtatott típusú objektumra
hívja meg az **equalTo()** metódust.

```
bool Bag::search(const Object* e, unsigned int &ind) const
{
    bool l = false;
    for(unsigned int i = 0; !l && i < _vect.size(); ++i){
        l = e->equalTo(_vect[i].v);
        ind = i;
    }
    return l;
};
```

Az eredetileg itt álló **e == _vect[i].v** feltétel nem lenne jó,
hiszen **e** és **_vect[i].v** eltérő memória címek.

bag.cpp

Fő program

```
ifstream f( "input.txt" );  
if(f.fail()){  
    cout << "Hiba a fájl nyitásakor!\n";  
    return 1;  
}
```

```
Bag h;  
char ch;  
while(f >> ch){  
    switch(ch) {  
        case 'i' : int n; f >> n;  
                h.putIn(new Integer(n));  
                break;  
        case 's' : string s; f >> s;  
                h.putIn(new String(s));  
                break;  
    }  
}  
try{  
    cout << "A megadott tomb leggyakoribb eleme: ";  
    if( h.maxElem()->is_Integer())  
        cout << (Integer*)h.maxElem()->value();  
    else if( h.maxElem()->is_String())  
        cout << (String*)h.maxElem()->value();  
}catch(Bag::Exceptions ex){  
    if(ex==Bag::EmptyBag) cout << "Ures a zsak! ";  
}
```

input.txt:

```
i 2  
i -4  
s alma  
s str  
i 13
```

A new helyet foglal egy Integer típusú objektumnak, elhelyezi benne az n értékét, és visszaadja a helyfoglalás címét.

típus kényszerítés: ha a h.MaxElem() egy Integer típusú objektum címe, azt (**Integer***)h.MaxElem() alakban írhatjuk, és erre már hívható a value() metódus.

main.cpp