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!

Ezt a feladatot többféleképpen is meg lehet oldani:

- Procedurális megoldás: maximum kiválasztás és számlálás
  - Megszámolhatjuk akár a számsorozat elemeinek a számsorozatbeli gyakoriságát (sőt elég egy elemnek a számsorozat hátralevő részében való előfordulásait megszámolni), akár a 0..m intervallum elemeinek a számsorozatbeli gyakoriságát.
  - A számlálások eredményeit eltárolhatjuk egy segéd-tömbben, hogy majd ezen hajtsuk végre maximum kiválasztást, vagy beágyazhatjuk a számlálást a maximum kiválasztásba.
- Objektumelvű megoldás: egy olyan tároló objektumban helyezzük el a számsorozat elemeit, amelytől lekérdezhető a benne található leggyakoribb elem. Legyen az elhelyezés is, és a lekérdezés is gyors.
  - Használhatunk tetszőleges elemeket tároló objektumot, vagy kifejezetten csak a 0..m intervallumbeli természetes számokat tároló objektumot.

Elemzés

Ezek a feladat változói, amelyek egyben a megoldást adó program változói is lesznek.

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

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

x' legalább 1 hosszú sorozat, x' elemei 0 és m' közé esnek

$Ef : m = m' \wedge x = x' \wedge |x| \geq 1 \wedge \forall i \in [1 .. |x|]: x_i \in [0 .. m]$

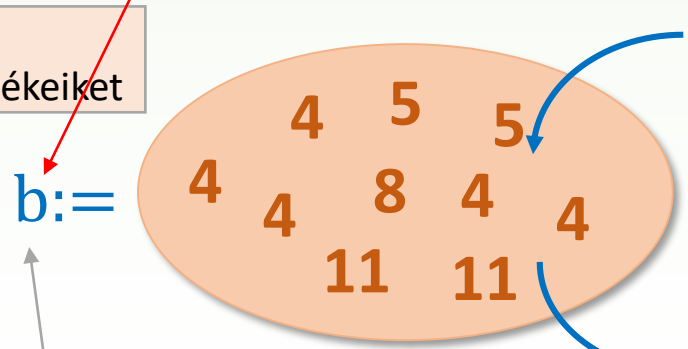
A változók értékét jellemzi a feladatot megoldó program elindulása előtt.

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

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

A változók értékét jellemzi a feladatot megoldó program leállása után.

az input-változók megőrzik kezdőértékeiket



betesz(b, e)

másképpen: $b:=b\cup[e]$
homogén binér művelet, ahol [e] az e-t tartalmazó egyelemű zsákot jelöli, neutrális eleme az üres zsák

e:=leggyakoribb(b)

Zsák típusú segédváltozó

Tervezés

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

Ef : $m = m' \wedge x = x' \quad \wedge \quad |x| \geq 1 \quad \wedge \quad \forall i \in [1 .. |x|]: x_i \in [0 .. m]$

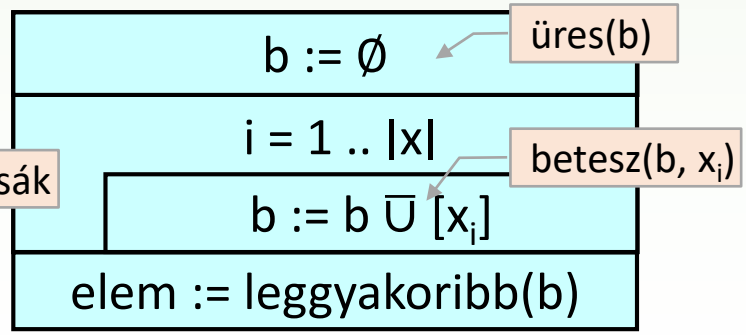
Dobáljuk be x' elemeit a zsákba, majd kérjük le annak a leggyakoribb elemét.

Uf : $m = m' \wedge x = x' \quad \wedge \quad b:Zsák \quad \wedge \quad b = \overline{\bigcup}_{i=1}^{|x|} [x_i] \quad \wedge \quad elem = leggyakoribb(b)$

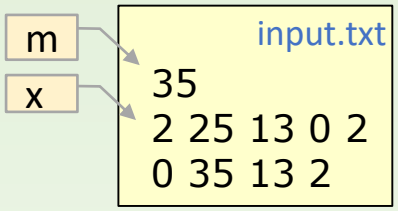
Az elemzés és a tervezés közötti határ (mi a feladat és hogyan oldjuk meg) elmosódik: végrehajtható specifikáció

A bezsákolás egy speciális összegzés:

$s = \Sigma_{i=m..n} f(i)$		~	$b = \overline{\bigcup}_{i=1..\|x\|} [x_i]$
művelet:	H, +, 0	~	Zsák, $\overline{\cup}$, Ø
felsorolt értékek:	$f(i)$	~	$[x_i]$
felsorolt indexek:	$i \in [m .. n]$	~	$i \in [1 .. \|x\|]$
eredmény:	s	~	b



Megvalósítás



input adatfolyam
#include <fstream>

```
ifstream f( "input.txt" );
if( f.fail() ) cout << "File open error!\n";
else {
        int m; f >> m;
        vector<int> x;
        int e;
        while( f >> e ){
                x.push_back(e);
        }
        ...
```

objektum-orientált metódushívás

egészeket tartalmazó sorozat
#include <vector>

```
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

üres(b)

Nem építjük fel az x sorozatot: közvetlenül a fájl elemeivel töltjük fel a zsákot.

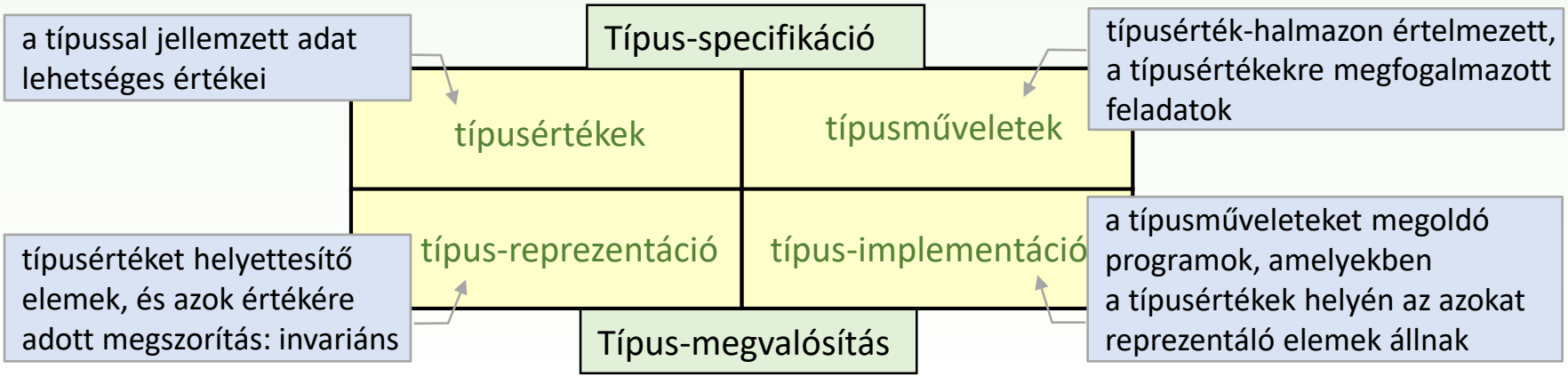
betesz(b, e)

leggyakoribb(b)

hátra van még a Zsák típus tervezése és kódolása

Adattípus fogalma

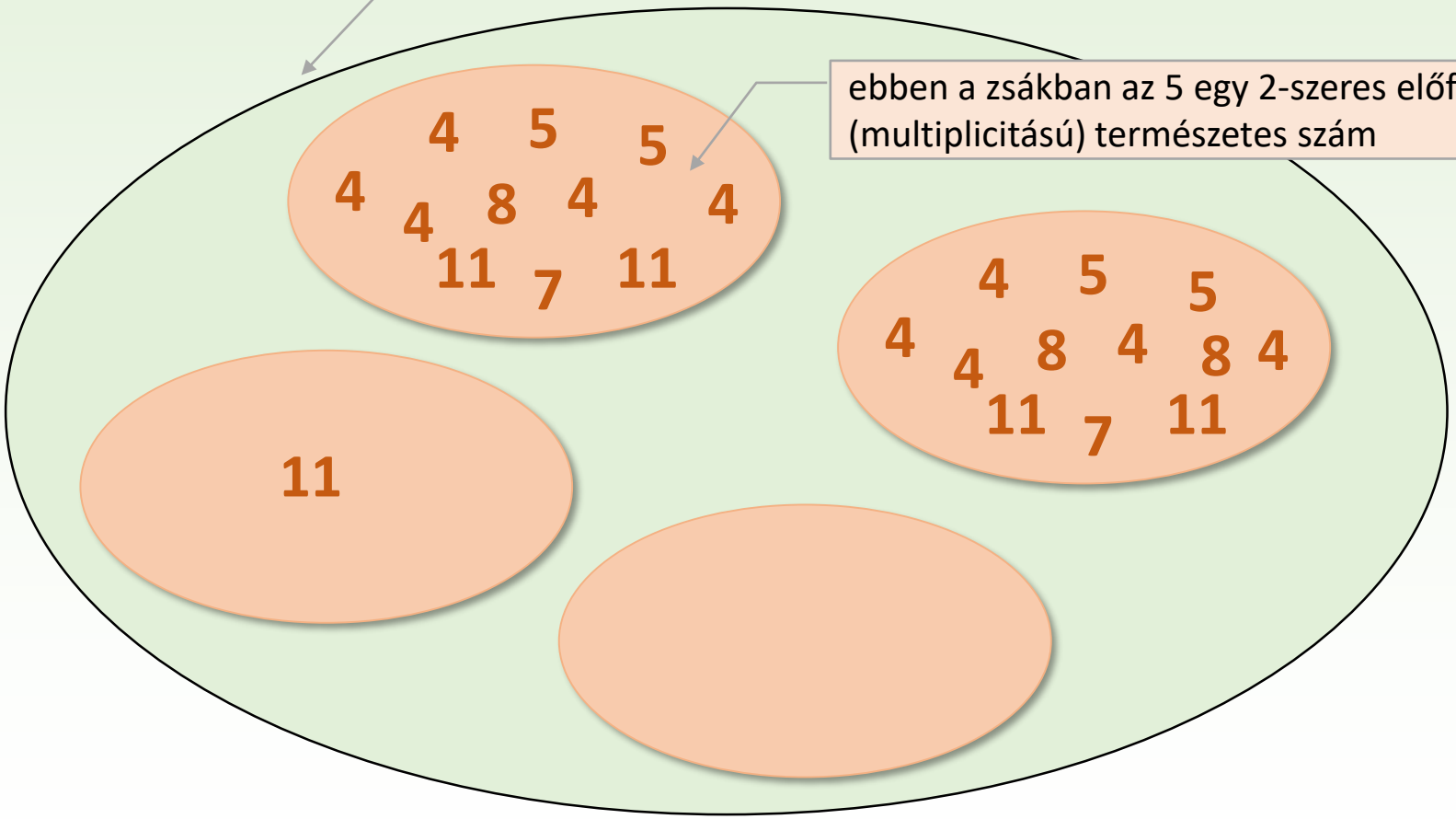
- Egy adat (változó) típusának definiálásához (tervéhez) szükség van a típus specifikációjára és annak megvalósítására.
- A típus-specifikáció megadja:
  - az adat által felvehető értékek halmazát
  - a típusértékekkel végezhető műveleteket
- A típus-megvalósítás megmutatja:
  - hogyan ábrázoljuk (reprezentáljuk)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



Zsák típus típusérték-halmaza

Egy természetes számokat tároló zsák típusú adatnak (változónak) egy értéke (típusérték) egy természetes számokat tároló zsák. Az összes ilyen zsák alkotja a zsák típus típusérték-halmazát.

ebben a zsákban az 5 egy 2-szeres előfordulású (multiplicitású) természetes szám



Zsák típus műveletei

Kiüríti a zsákot:

üres(b) b:Zsák

Betesz egy elemet a zsákba:

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

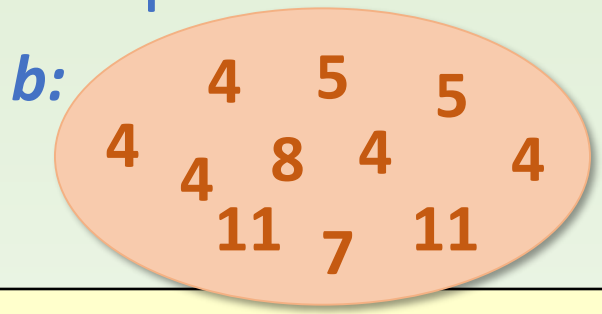
betesz(b, e) b:Zsák, e:ℕ

Zsák leggyakoribb eleme:

adjon hibajelzést, ha a zsák üres

e:=leggyakoribb(b) b:Zsák, e:ℕ

Zsák típus reprezentációja



A zsákban csak 0..m közötti egészek lehetnek, amelyek előfordulási gyakoriságait egy tömbben eltárolhatjuk.

	0	1	2	3	4	5	6	7	8	9	10	11	…	m
vek:	0	0	0	0	5	2	0	1	1	0	0	2	…	0

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

max: 4 : $\mathbb{N}$

típusinvariáns:

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

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

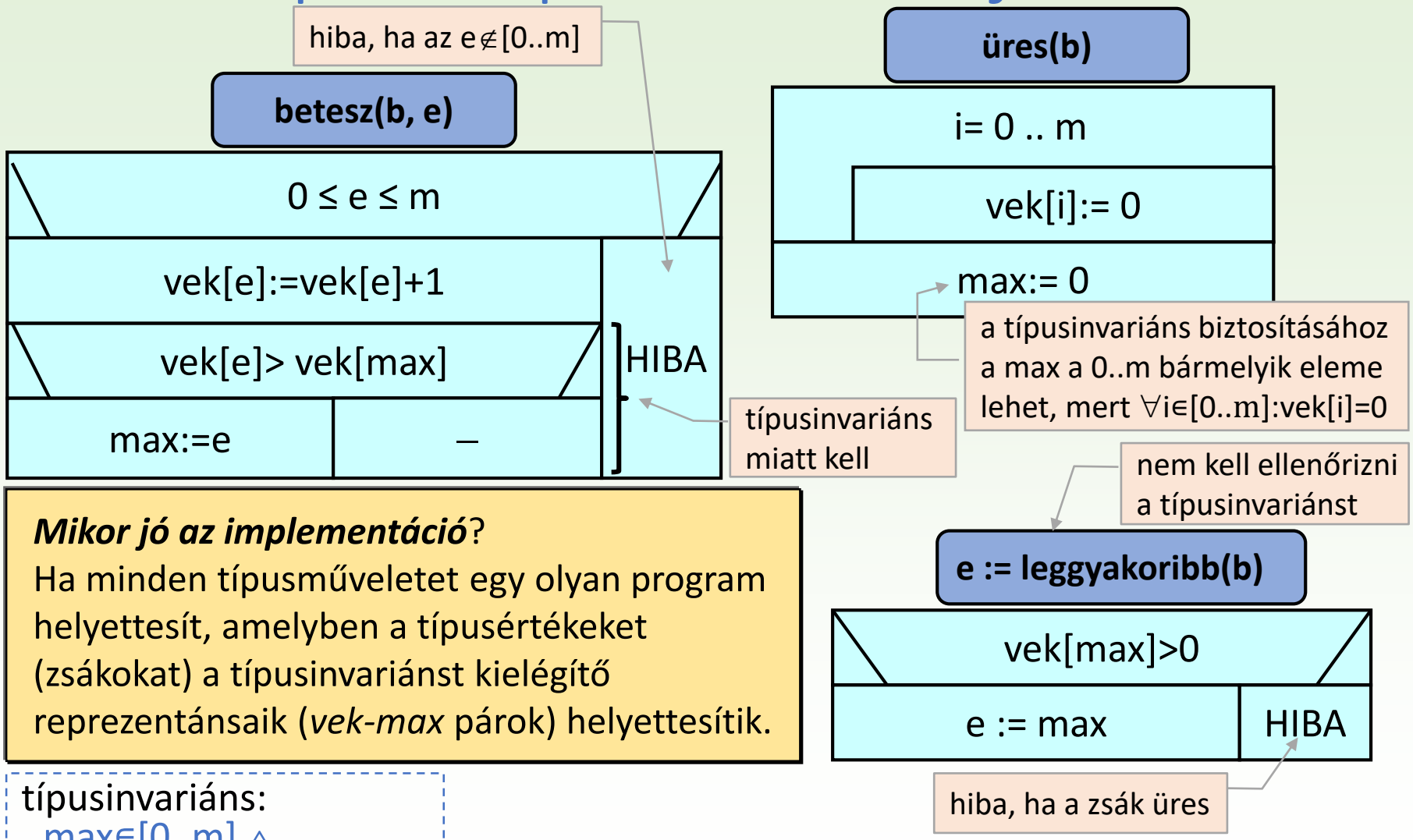
reprezentáló adatok tulajdonságai és a közöttük levő kapcsolatok

külön nyilvántartjuk a leggyakoribb elemet

hasznos melléktermék: $vek[max]=0$ jelzi, hogy a zsák üres

Mikor jó egy reprezentáció?
Ha minden típusértéket (zsákot) kifejezhetünk egy típusinvariánst kielégítő reprezentánssal (*vek-max* párral), valamint
minden típusinvariánst kielégítő reprezentáns (*vek-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 (*vek-max* párok) helyettesítik.

típusinvariáns:
$max \in [0..m] \wedge$
$vek[max] = \max_{i=0}^{m} vek[i]$

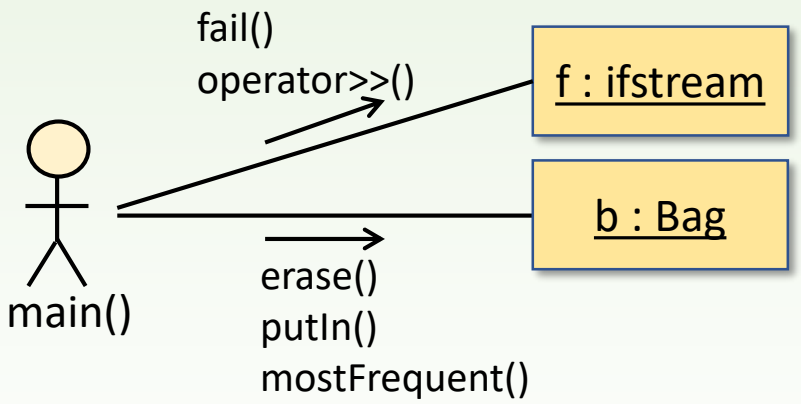
Objektum fogalma

például egy zsák

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

üres(), betesz(), leggyakoribb()

vek, max



1

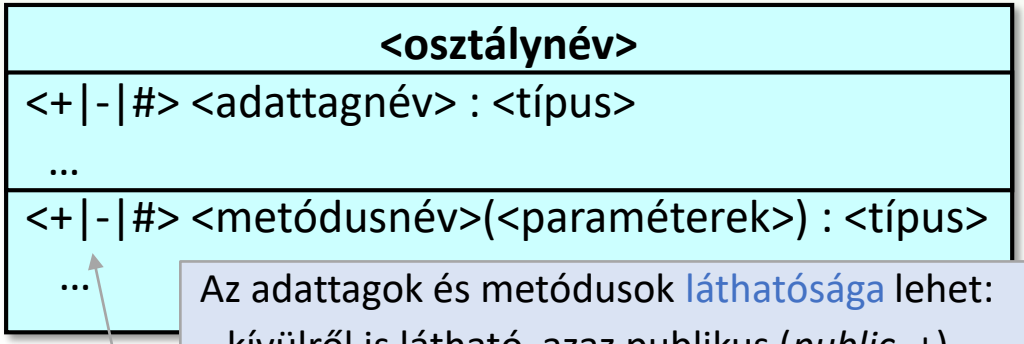
Az objektum-orientáltság lényeges ismérve az egységbezárás: egy adott feladatkör megvalósításához szükséges adatokat és az azokat manipuláló programrészeket a program többi részétől elkülönített egységként adjuk meg.

Osztály fogalma és UML jelölése

- Az osztály egy objektum szerkezetének és viselkedésének a mintáját adja meg, azaz
  - felsorolja az objektum adattagjait (azok nevét és típusát)
  - megadja az objektumra meghívható metódusokat (azok nevét, paraméterlistáját, visszatérési értékének típusát)
- Az osztály lényegében az objektum típusa: az objektumot az osztálya alapján hozzuk létre, azaz példányosítjuk.
- Egy osztályhoz több objektum is példányosítható: minden objektum olyan saját adattagokkal és metódusokkal rendelkezik, amelyeket az osztályleírás megad.

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

üres() : void,
betesz(int) : void,
leggyakoribb() : int



Az adattagok és metódusok láthatósága lehet:
- kívülről is látható, azaz publikus (*public* +)
- külvilág elől rejtett: privát (*private* -) vagy védett (*protected* #)

Típus és Osztály

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

Zsák típus:

típus-specifikáció

típus-értékek

zsákok

típus-műveletek

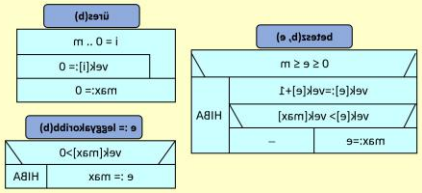
üres(b)
betesz(b, e)
e:=leggyakoribb(b)
b:Zsák, e:ℕ

reprezentáció

vek : $\mathbb{N}^{0..m}$
max : ℕ

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

implementáció



típus-megvalósítás

Zsák osztály:

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

Zsák
- vek : int[0..m] - max : int
+ üres() : void + betesz(e:int) : void + leggyakoribb() : int

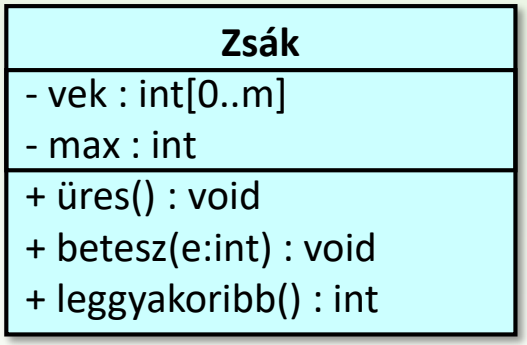
for i=0..m **loop** vek[i] := 0 **endloop**
max := 0

if vek[max]>0 **then return** max
else error
endif

if 0≤e≤m **then**
++vek[i]
if vek[e]>vek[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.

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]++;
}
```

Amikor egy objektum egy tagjára hivatkozunk, akkor magát az objektumot is ismernünk kell. Ezért az objektumra hivatkozó változót fel kell tüntetni a hivatkozásban a tag előtt.

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

Egy objektum rejtett (privát, védett) tagjait csak az objektum metódusainak definíciójában (annak törzsében) használhatjuk, máshol ezek illegális hivatkozásnak számítanak.

2

Az objektum orientáltság fontos ismérve az elrejtés: amikor az egységbe zárt elemek láthatóságát korlátozzuk. (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

A metódus törzsében szereplő, az objektum megjelölése nélkül meghivatkozott objektum-tag ahhoz az objektumhoz tartozik, amelyikkel a metódust meghívják.

Konstruktor

Egy objektum példányosításakor mindig egy speciális metódus, a konstruktor hívódik meg, amelyik sorban egymás után létrehozza 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);
    …
};
```

Az üres konstruktor hatására a *Bag b* deklaráció egy nulla hosszú _vec nevű sorozatot, és egy _max nevű integert hozna létre.

Kell egy olyan paraméteres konstruktor, amellyel beállítható a reprezentációban használt _vec hossza. Pl.: *Bag b(35)*

Először létrehozza az adattagokat (ha mást nem mondunk) azok üres konstruktoraival, majd végrehajtja a törzse utasításait.

```
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

```
#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;
}
```

Bag osztály definíció bemásolása

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

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, és kiegészítené azokat a getArray() metódussal. A tesztelést a Bag osztály helyett erre a Bag_Test osztályra végeznénk.

„inline” definíció