

Programozás alapjai II.
(10. ea) C++
hibakezelés és STL bevezető

Szeberényi Imre, Somogyi Péter

BME IIT

<szebi@iit.bme.hu>



MŰEGYETEM 1782

Hol tartunk?

- OO alapismeretek, paradigmák
 - egységbezárás (encapsulation)
 - osztályok (adatszerkezet + műveletek)
 - többarcúság (polymorphism)
 - műveletek paraméter függőek, tárgy függőek (kötés)
 - példányosítás (instantiation)
 - öröklés (inheritance)
 - generikus adatok és algoritmusok
- Konkrét C++ szintaxis

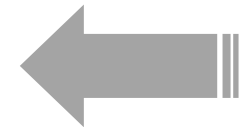
Konstruktor feladatai (ism)

- Öröklési lánc végén hívja a virtuális alaposztályok konstruktorait.
- Hívja a közvetlen, nem virtuális alaposztályok konstruktorait.
- Létrehozza a saját részt:
 - beállítja a virtuális alaposztály mutatóit
 - beállítja a virtuális függvények mutatóit
 - hívja a tartalmazott objektumok konstruktorait
 - végrehajtja a programozott törzset



Destruktor feladatai (ism)

- Megszünteti a saját részt:
 - végrehajtja a programozott törzset
 - tartalmazott objektumok destruktoraik hívása
 - virtuális függvénymutatók visszaállítása
 - virtuális alaposztály mutatóinak visszaállítása
- Hívja a közvetlen, nem virtuális alaposztályok destruktoraik.
- Öröklési lánc végén hívja a virtuális alaposztályok destruktoraik.



https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_09 → ctor_dtor

https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_10 → ctor_dtor2

Ma

- Kivételkezelés
 - működés részletei,
 - hatása az objektumok élettartamára
 - Konstruktorban, destruktorban
- Létrehozás/megsemmisítés működésének felhasználása (pl. fájlkezeléshez)
- STL bevezető (áttekintés)

Erőforrás foglalás-felszabadítás

- Gyakori, hogy erőforrásként kell kezelni valamit (memória, fájl, eszköz, stb.):
 - lefoglalás
 - feldolgozás
 - felszabadítás
- Az ilyen esetekben külön figyelmet kell fordítani arra, hogy a feldolgozás közben észlelt hiba esetén is gondoskodjunk a felszabadításról.

Erőforrás foglálás-felszabadítás/2

```
FILE *fp = fopen("f1.txt", "r");    // megnyitás

try {
    // file feldolgozása
} catch (...) {
    fclose(fp);    // lezárás hiba esetén
    throw;        // tovább
}
fclose(fp);    // lezárás normál esetben
```

Erőforrás foglálás-felszabadítás/3

```
class FILE_ptr {  
    FILE *p;  
public:  
    FILE_ptr(const char *n, const char *m) {  
        p = fopen(n, m); }  
    ~FILE_ptr { fclose(p); }  
    operator FILE*() { return p; }  
};
```

cast operator

```
{  
    FILE_ptr fp("f1.txt", "r");  
    fprintf(fp, "Hello");  
}
```

automatikus cast

destruktor megszüntet (bezár)

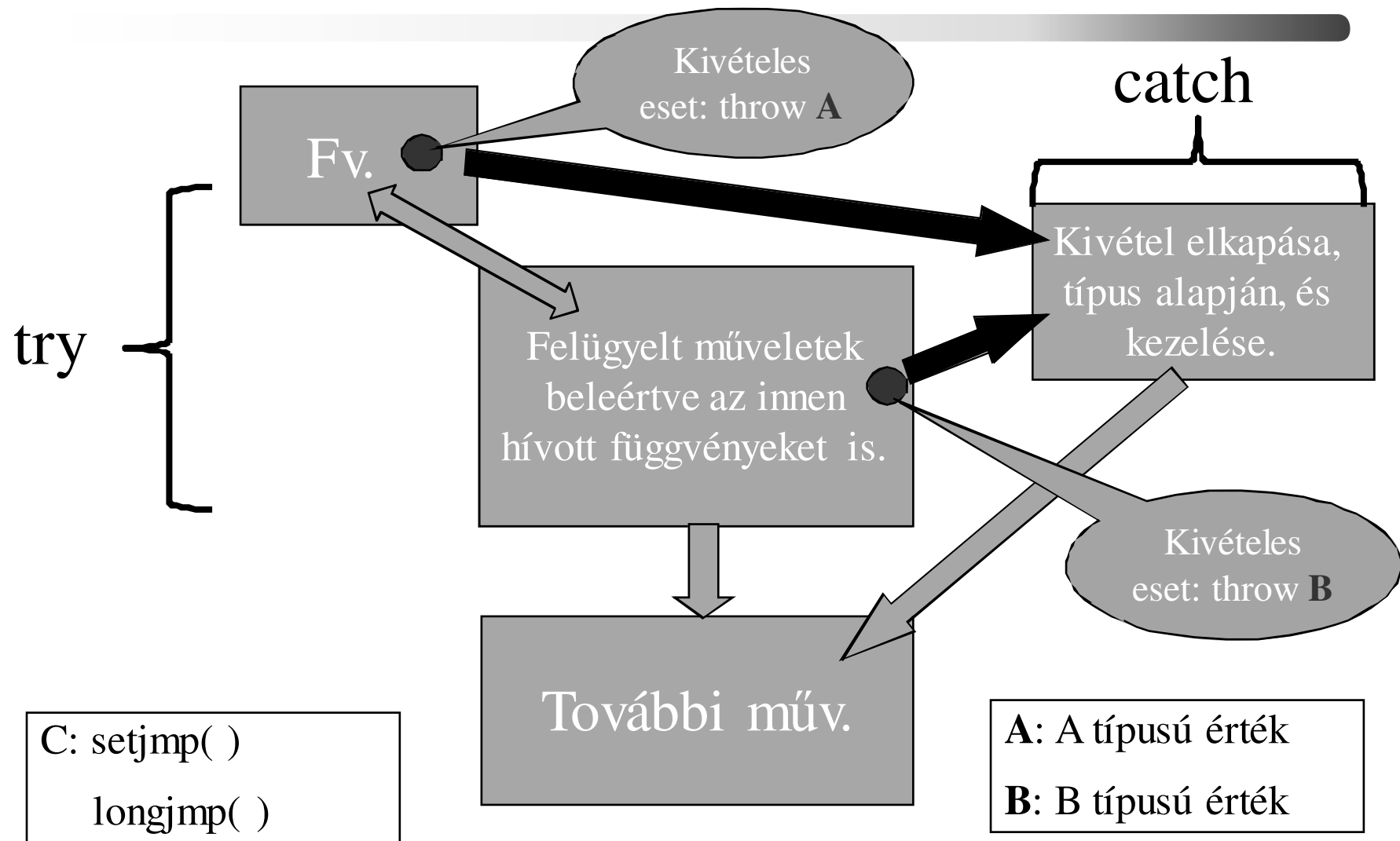
Kivételes esetek kezelése (ism.)

- Kinek kell jelezni?
 - felhasználó, másik programozó, másik program
 - saját magunknak
- A kivételes eset kezelése gyakran nem annak keletkezési helyén történik.
(Legtöbbször nem tudjuk, hogy mit kell tenni. Megállni, kiírni valami csúnyát, stb.)

Kivétel kezelés (ism.)

- C++ típus orientált kivételkezelést támogat, amivel a kivételes esetek kezelésének szinte minden formája megvalósítható.
- A kivételkezeléshez tartozó tevékenységek:
 - figyelendő kódrészlet kijelölése (try)
 - kivétel továbbítása (throw)
 - esemény lekezelése (catch)

Kivételkezelés = globális goto (ism)



Kivételkezelés/2 (ism)

```
try {
```

..... Kritikus művelet1 pl. egy függvény hívása, aminek a belsejében: `{if (hiba) throw kifejezés_tipus1;}`

..... Kritikus művelet2 pl. egy másik függvény hívása, aminek a belsejében: `{if (hiba) throw kifejezés_tipus2;}`

```
} catch (típus1 param) {
```

..... Kivételkezelés1

```
} catch (típus2 param) {
```

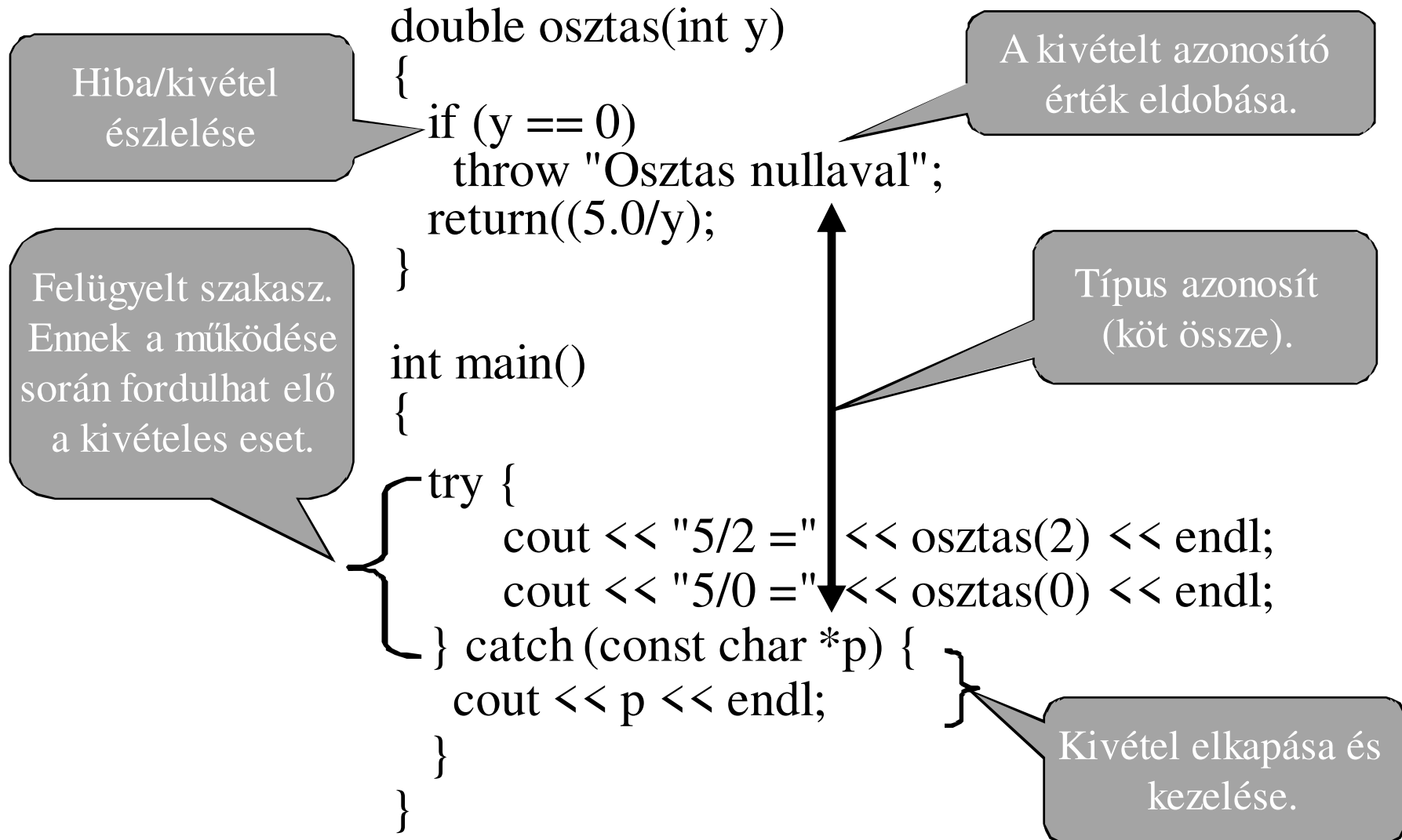
..... Kivételkezelés2

```
}
```

A hiba tetszőleges mélységben keletkezhet.

Közvetlenül a **try-catch** blokkban a **throw**-nak nincs sok értelme, hiszen akkor már kezelni is tudnánk a hibát.

Kivételkezelés példa (ism)



Újdonságok a korábbiakhoz

- A dobott kivétel alap ill. származtatott objektum is lehet.

```
try {  
    throw E();  
} catch(H) {  
    // mikor jut ide ?  
}
```

1. **H és E azonos típusú,**
2. **H bázisosztálya E-nek,**
3. **H és E mutató és teljesül rájuk 1. vagy 2.,**
4. **H és E referencia és teljesül rájuk 1. vagy 2.**

Következtetések

- Célszerű kivétel osztályokat alkalmazni, (pl. `std::exceptions`) amiből származtatással újabb kivételeket lehet létrehozni.
- A dobás értékparamétert dob, ezért az elkapáskor számolni kell az alaposztályra történő konverzióval (adatvesztés).
 - ➔ Célszerű pointert, vagy referenciát alkalmazni.
 - ➔ Kell másoló konstruktor.

Kivételek specifikálása

- Függvény deklarációsakor/definíciójakor megadható, hogy milyen[†] kivételeket generál az adott függvény.
- Ha mást is generálna, akkor az automatikusan meghívja az `unexpected()`[†] handlert.

`void f1() throw (E1, E2);† // csak E1, E2`

`void f2() throw();† // semmi`

`void f3(); // bármi`

`void f3() noexcept(true) // semmi (C++11-től)`

[†] C++17-től megszűnik a függvények din. kivétel specifikációja. Csak dob/nem dob.

Továbbdobás


```
try {  
    throw E();  
} catch(H) {  
    if (le_tudjuk kezelni) {  
        ....  
    } else {  
        throw;  
    }  
}
```

Az eredeti dobódik tovább,
nem csak az elkapott
változat.

Paraméter nélkül

Minden elkapása

```
try {  
    throw E();  
} catch(...) {  
    // szükséges feladatok  
    throw;  
}
```



Minden kivétel

A kezelők sorrendje fontos!

Rollback (*stack unwinding*)

```
try {  
    A a;  
    B b;  
    C *cp = new C;  
    if (hiba) throw "Baj van";  
    delete cp;  
} catch(const char *p) {  
    // A létezik ?  
    // B létezik ?  
    // *cp által mutatott obj. létezik ?  
}
```

Minden a blokkban deklarált, "létező" objektum destruktora meghívódik.

Hiba esetén C példánya nem szabadul fel, de cp megszűnik.

Melyik obj. létezik ?

- Csak az az objektum számít létezőnek, amelynek a konstruktora lefutott.
- Ha a konstruktor nem fut le, akkor a rollback során a destruktor sem fog végrehajtódni.
- Előző példában C konstruktora lefutott ugyan, de nem deklarációval hoztuk létre, hanem dinamikusán.

https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_10 → kodreszletek

Kivétel a konstruktorban

- Lényegében a kivételkezelés az egyetlen mód arra, hogy a konstruktor hibát jelezzon.
- Hiba esetén gondoskodni kell a megfelelő obj. állapot előállításáról.

Inicializáló listán keletkező kivétel elfogása:

```
struct A {  
    B b;  
    A() try  
        :b() { // konstruktor programozott része  
    } catch (...) {  
        ... // kivételkezelés  
        // ha eljut idáig itt továbbdob  
    }  
};
```

Kivétel a destruktorbán

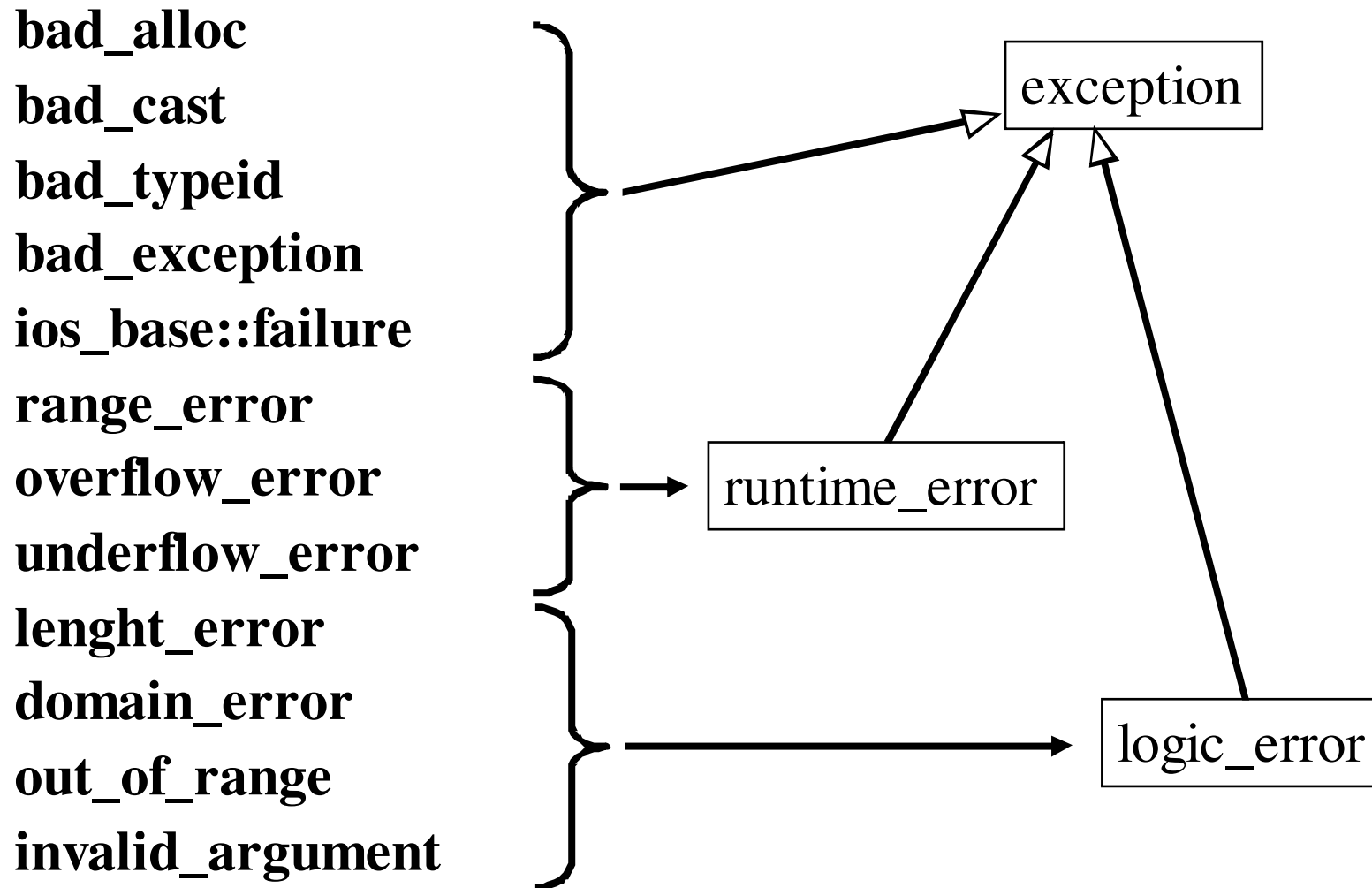
Destruktor hívás oka:

1. Normál meghívás
2. Kivételkezelés (rollback) miatti meghívás.
Ekkor a kivétel nem léphet ki a destruktorból.

Destruktorban keletkező kivétel elfogása:

```
A::~~A() try {  
    // destruktor programozott törzse  
} catch (...) {  
    ... // kivételkezelés  
    // ha eljut idáig, továbbdob  
}
```

Szabványos kivételek (stdexcept)



Szabványos kivételek/2

```
class exception {
    ...
public:
    exception() throw();
    exception(const exception&) throw();
    exception& operator=(const exception&) throw();
    virtual ~exception() throw();
    virtual const char *what() const throw();
};
class runtime_error : public exception {
public:
    explicit runtime_error(const string& what_arg);
};
class logic_error : public exception {
public:
    explicit logic_error(const string& what_arg );
};
```


Szabványos kivételek/3

- A standard könyvtár nem bővíti az *exception* osztály függvényeit, csak megfelelően átdefiniálja azokat.
- A felhasználói programnak nem kötelessége az *exception*-ből származtatni, de célszerű.

```
try {  
    .....  
  
} catch (exception& e) {  
    cout << "expectionból származik"  
    cout << e.what() << endl;  
} catch (...) {  
    cout << "Ez valami más\n";  
}
```

referencia

Kapcsolódó függvények

- `std::terminate_handler`
`set_terminate(std::terminate_handler) throw();` 🕒
 - beállítja a terminate handlert
- `void terminate();`
 - meghívja a terminate handlert
- `bool uncaught_exception() throw();` 🕒
 - kivételkezelés folyamatban van, még nem talált rá a megfelelő handlerre (nem kapták el). Destruktorban lenne szerepe, de ...

🕒 C++11-től a szignatúra változik

Szabványos könyvtár (STL)

Általános célú, újrafelhasználható elemek:

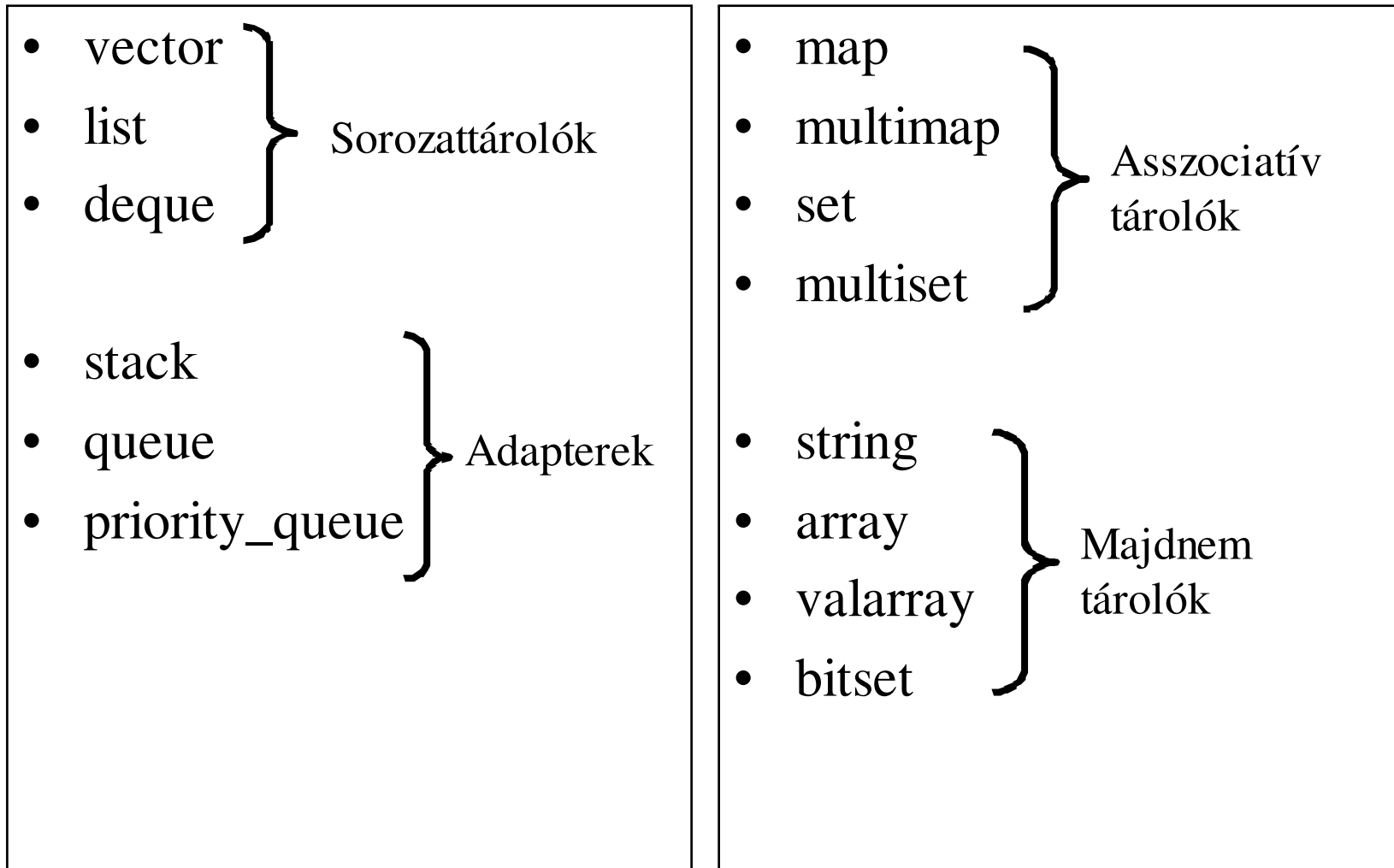
- tárolók, majdnem tárolók
- algoritmusok
- függvények
- bejárók
- kivételek
- memóriakezelők
- adatfolyamok

<http://www.sgi.com/tech/stl/>

<http://www.cppreference.com/cppstl.html>

<http://www.cplusplus.com/reference/stl/>

Szabványos tárolók



vector template

```
template <class T, class Allocator = allocator<T> >
class vector {
public:
// Types
    typedef T value_type;
    typedef Allocator allocator_type;
    class iterator;
    class const_iterator;
    typedef typename Allocator::size_type size_type;
    typedef typename Allocator::difference_type difference_type;
    typedef typename Allocator::reference reference;
    typedef typename Allocator::pointer pointer;
    typedef typename std::reverse_iterator<iterator> reverse_iterator;

    ....

```

vector template/2

```
// Construct/Copy/Destroy  
explicit vector(const Allocator& = Allocator());  
explicit vector(size_type, const Allocator& = Allocator ());  
vector(size_type, const T&, const Allocator& = Allocator());  
vector(const vector<T, Allocator> &);  
vector<T,Allocator> & operator=(const vector<T, Allocator> &);  
template <class InputIterator>  
    void assign(InputIterator start, InputIterator finish);  
void assign(size_type, const);  
allocator_type get_allocator () const;
```

vector template/3

// Iterators

iterator begin();

const_iterator begin() const;

iterator end();

const_iterator end() const;

reverse_iterator rbegin();

const_reverse_iterator rbegin() const;

reverse_iterator rend();

const_reverse_iterator rend() const;

vector template/4

// Capacity

```
size_type size() const;  
size_type max_size() const; // elméleti maximum  
void resize(size_type);  
void resize(size_type, T);  
size_type capacity() const; // jelenleg allokált  
bool empty() const;  
void reserve(size_type);
```


vector template/5

// Element Access

```
reference operator[](size_type);  
const_reference operator[](size_type) const;  
reference at(size_type);  
const_reference at(size_type) const;  
reference front();  
const_reference front() const;  
reference back();  
const_reference back() const;
```

vector template/6

// Modifiers

void push_back(const T&);

void pop_back();

iterator insert(iterator, const T&);

void insert(iterator, size_type, const T&);

template <class InputIterator>

void insert(iterator, InputIterator, InputIterator);

iterator erase(iterator); iterator erase(iterator, iterator);

void swap(vector<T, Allocator> &);

void clear()

};

Tárolók által definiált típusok

Általános, minden tárolóra érvényes

- value_type, allocator_type,
- reference, const_reference,
- pointer, const_pointer
- iterator, const_iterator
- reverse_iterator, const_reverse_iterator
- difference_type
- size_type

Minden asszociatív tárolóra érvényes

- key_type
- key_compare

Tárolók műveletei (áttekintés)

Általános, minden tárolóra érvényes

- létrehozás/megszüntetés: konstruktorok/destruktor
- értékadás: operator=, assign()
- iterátor példányosítás: begin(), end(), rbegin(), reend()
- méret lekérdezés: size(), max_size(), capacity() empty()
- módosítók: insert(), erase(), clear(), swap()

Lista kivételével minden sorozattárolóra érvényes

- elemek elérése: front(), back(), operator[], at()
- módosítók: push_back(), pop_back()

Minden asszociatív tároló érvényes

- count(), find(), lower/upper_boud(), equal_range()

Speciális, csak egy adott tárolóra alkalmazható pl:

- elemek elérése: pop_front(), push_front(),
- módosítók: merge(), splice(), uniq(), remove(), sort(), ...

Létrehozás, iterátorok

vektor<double> dvec;

vektor<int> ivec(10);

vector<char> cvec(5, '*');

int v[] = {1, 2, 3}; vector<int>

iv(v, v+3);

vector<int> iv2(iv);

konstruktorok, destr.:

- container()
- container(n)
- container(n,x)
- container(first, last)
- container(c)
- ~container()

vektor<int>::iterator it;

vector<int>::reverse_iterator rit;

it = ivec.begin();

rit = ivec.rend();

iterátor:

- container::iterator
- container::reverse_iterator
- begin(), end()
- rbegin(), rend()

```
for (it = ivec.begin(); it != ivec.end(); ++it)
    std::cout << *it;
```

Értékadás, elérés, size

```
template <typename C> // Segédsablon a példákhoz
void print(C& co, const char* sep = ",") {
    for (typename C::iterator it = co.begin(); it != co.end(); ++it )
        std::cout << *it << sep;
}
```

```
list<int> ilist1, ilist2(100);
ilist1.assign(5, 3);
ilist2.assign(ilist1.begin(), ilist1.end());
cout << ilist2.size(); // 5
print(ilist2, ", " ); // 3, 3, 3, 3, 3,
vector<int> ivec(ilist2.begin(), ilist2.end());
ivec[0] = 6;
cout << ivec.front(); // 6
cout << ivec.back(); // 3
```

assign: (2 fajta)
• assign(n,x)
• assign(first, last)

Módosítók

```
template <typename C> // Segédsablon a példákhoz
void reverse_print(C& co, const char* sep = ",") {
    for (typename C::reverse_iterator it = co.rbegin();
        it != co.rend(); ++it )
        std::cout << *it << sep;
}
```

```
char duma[] = "HELLO";
vector<char> cv(duma, duma+5);
reverse_print(cv, ""); // OLLEH
cv.insert(cv.end(), ' ');
cv.insert(cv.end(), duma, duma+5);
print(cv, ""); //HELLO HELLO
cv.insert(cv.begin(), 3, '*');
print(cv, ""); //***HELLO HELLO
```

insert: (3 fajta)

- insert(pos, x);
- insert(pos, first, last);
- insert(pos, n, x)

További módosítók

```
deque<int> iq1, iq2;
iq1.push_back(3); iq1.push_back(4);
iq1.push_back(8);iq1.push_back(10);
iq1.push_back(2);iq1.push_back(7);
cout << iq1.front();           // 3
print(iq1);                     // 3,4,8,10,2,7,
iq1.pop_back();
print(iq1);                     // 3,4,8,10,2,
iq1.erase(iq1.begin()+1);
print(iq1);                     // 3,8,10,2,
iq1.erase(iq1.begin()+1, iq1.end());
print(iq1);                     // 3,
```

erase: (2 fajta)

erase(pos);

erase(first, last);

https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_10 → kodreszletek

Asszociatív tárolók műveletei

```
set<int> set1;  
set1.insert(4); set1.insert(3);  
cout << set1.count(3); // 1  
set1.insert(3); set1.insert(10);  
print(set1); // 3,4,10,  
if (set1.find(4) != set1.end())  
    cout << "megvan"; // megvan  
typedef set<int>::iterator myit;  
myit it = set1.lower_bound(4);  
cout << *it; // 4  
it = set1.upper_bound(4);  
cout << *it; // 10  
std::pair<myit, myit> rit;  
rit = set1.equal_range(4);  
cout << *rit.first << ":" << *rit.second; // 4:10
```

- count(x)
- find(x)
- lower_bound(x)
- upper_bound(x)
- equal_range(x)

vector<T, Alloc>

Speciális műveletek:

- `capacity()`
- `reserve()`
- `resize(n)`, `resize(n, val)`

```
int v[] = {0, 1, 2};  
vector<int> iv(v, v+3);  
iv.push_back(3);  
iv.at(5) = 5; // hiba
```

Nincs:

- `push_front`, `pop_front`
- asszociatív op.

```
iv.resize(7, -8);  
iv.at(5) = 5; // nincs hiba  
print(iv) // 0,1,2,3,-8,5,-8,
```

list<T, Alloc>

Speciális műveletek:

- merge(list),
- merge(list, bin_pred)
- remove(val)
- remove_if(un_pred)
- resize(n), resize(n, val)
- sort(), sort(bin_pred)
- splice(p, list)
- splice(p, list, first)
- splice(p, list, first, last)
- unique(), unique(bpred)

Nincs:

- at(), operator[]()
- asszociatív op.

```
list<int> il(2, -3);
il.push_front(9);
il.push_back(2);
il.sort();
print(il); // -3, -3, 2, 9,
il.unique();
list<int> il2(3, 4);
il.merge(il2);
print(il); // -3, 2, 4, 4, 4, 9,
```

deque <T, Alloc>

Kétfélgű sor

Speciális műveletek:

- `resize(n)`, `resize(n, val)`

Nincs:

- asszociatív op.

```
deque<int> dq;  
dq.push_back(6);  
dq.push_front(9);  
print(dq); // 9, 6,  
dq.resize(6, -3);  
print(dq); //9, 6, -3, -3, -3,-3,
```

```
dq.back() = 1;  
print(dq); // 9, 6, -3, -3, -3, 1,  
dq[2] = 2;  
print(dq); // 9, 6, 2, -3, -3, 1,  
dq.at(3) = 0;
```

```
if (!dq.empty())  
    print(dq); // 9, 6, 2, 0, -3, 1,
```

stack<T, deque>

Elrejtí a kétvégű sor nem verem stílusú műveleteit.

Műveletek:

- empty()
- push()
- pop()
- top()
- stack()
- stack(cont)

```
stack<int> s;  
s.push(1);  
s.push(2);  
s.push(3);  
s.top() = 4;  
s.push(13);
```

```
while (!s.empty()) {  
    cout << s.top() << ", "; s.pop();  
} // 13, 4, 2, 1,
```

queue <T, deque>

Elrejtí a kétvégű sor nem sor stílusú műveleteit.

Műveletek:

- `empty()`
- `push()` → `push_back()`
- `pop()` → `pop_front()`
- `front()`
- `back()`
- `queue()`, `queue(cont)`

```
queue<int> q;  
q.push(1);  
q.push(2);  
q.push(3);  
q.back() = 4;  
q.push(13);
```

```
while (!q.empty()) {  
    cout << q.front() << ", "; q.pop();  
} // 1, 2, 4, 13,
```

priority_queue <T, vector, Cmp>

Prioritásos sor. Alapesetben a < operátorral hasonlít.

Műveletek:

- empty()
- push()
- pop()
- top()
- priority_queue()

```
priority_queue<int> pq;  
pq.push(1);  
pq.push(2);  
pq.push(3);  
pq.push(-2);  
pq.push(13);
```

```
while (!pq.empty()) {  
    cout << pq.top() << ", "; pq.pop();  
} // 13, 3, 2, 1, -2,
```

map < Key, T, Cmp, Alloc >

Asszociatív tömb

- (kulcs, érték) pár tárolása
- alapértelmezés szerint < operátorral hasonlít
- map maga is összehasonlítható

```
map<string, int> m;  
m["haho"] = 8;  
m["Almas"] = 23;  
cout << m["haho"] << endl;  
cout << m["Almas"] << endl;  
map<string, int>::iterator i = m.find("haho");
```


pair < const Key, mapped_type >

Párok

- map bejárásakor párok (pair) sorozatát kapjuk
- A kulcsra first, az értékre second mezővel hivatkozhatunk

```
map<string, int> m;  
m["haho"] = 8; m["Almas"] = 23; m["xx"] = 13;  
map<string, int>::iterator p;  
for (p = m.begin(); p != m.end(); p++) {  
    cout << p->first << ": ";  
    cout << p->second << ", ";  
} // almas: 23, haho: 8, xx: 13
```

set<Key, Cmp, Alloc>

Halmaz

- olyan map, ahol nem tároljuk az értéket
- alapértelmezés szerint < operátorral hasonlít
- map-hoz hasonlóan összehasonlítható

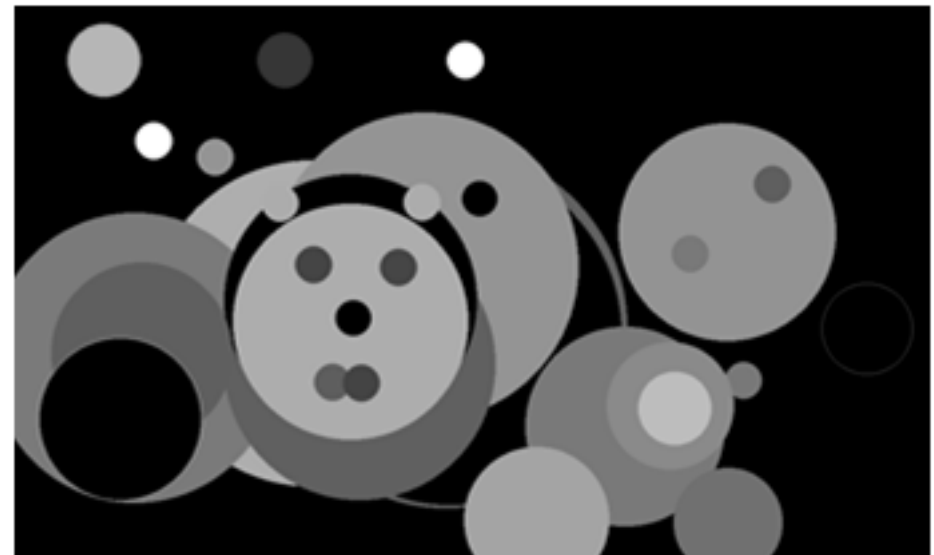
```
set<long> s;  
s.insert(3); s.insert(3);  
s.insert(7); s.insert(12); s.insert(8);  
cout << s.count(6) << endl; // 0  
cout << s.count(3) << endl; // 1  
set<long>::iterator i = s.find(3);  
print(s); // 3, 7, 8, 12,
```

Demo: Eseményvezérelt program

Demonstrációs cél: eseményvezérlés grafikus felhasználói felület működése, egyszerű SDL grafika, újrafelhasználható elemek, callback technika

Specifikáció: Eseményekre reagáló alakzatok (körök) felrakása a képernyőre. Vezérlő gombok (töröl, kilép) kialakítása.

Események: Egérmozgatás, kattintás, húzás (drag)

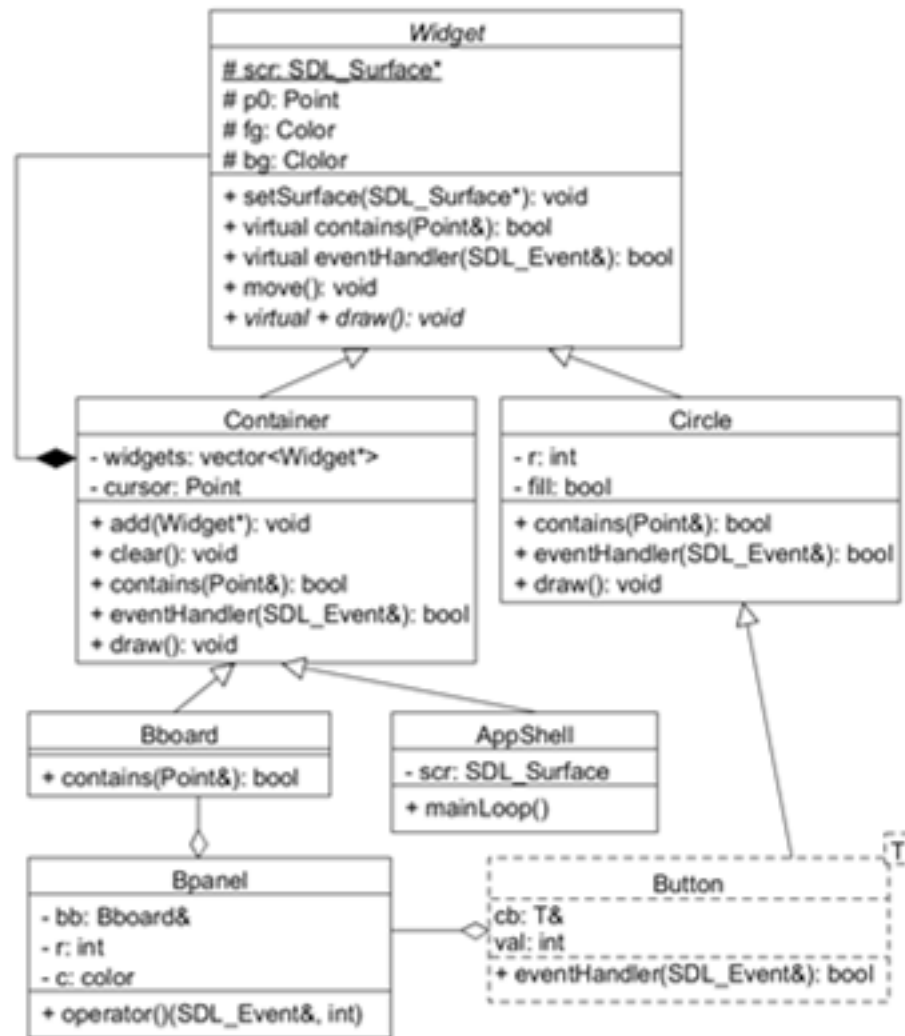


https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_10 → SDL_bborad

Szereplők

- grafikus primitívek:
 - Widget, Circle, Button, ActiveBg
- összetett grafikus elemek:
 - Container, BulletinBoard (Bboard),
 - Application Shell (AppShell)

Osztálydiagram



- A modell nem kezel ablakokat, így vágás sincs az ablakok határán.
- Minden grafikus elem a képernyő tetszőleges pontján megjelenhet.
- Egy esemény érkezésekor (pl. kattintás) a konténerbe helyezéssel ellentétes sorrendben megkérdezzük az objektumokat, hogy az adott koordináta hozzá tartozik-e (contains fv.).
- Amennyiben igen, akkor meghívjuk a eseménykezelőjét.
- Az eseménykezelő visszatérési érték jelzi, hogy kezelt-e.
- A Bboard objektum hazudós, mert minden pozícióra „rábólint”, hogy az övé. Így végigfut a tárolóban az ellenőrzés.

Widget

```
class Widget {
protected:
    static SDL_Surface *scr; // eldugott globális változó
    Point p0;                // Widget origója
    Color fg;                // Előtér szín
    Color bg;                // Háttérszín
public:
    Widget(const Point& p0 = Point(), const Color& fg = WHITE,
           const Color& bg = BLACK) :p0(p0), fg(fg), bg(bg) {}
    static void setSurface(SDL_Surface* s) { scr = s; }
    virtual bool contains(const Point& p) { return false; }
    virtual bool eventHandler(const SDL_Event& ev) {
        return false; };
    void move(const Point& d);
    virtual void draw() const = 0;
    virtual ~Widget() {}
};
```

SDL 2-ben Renderer
pointert tárolunk a Surface
helyett (ld. mintaprogram)

Circle

```
class Circle : public Widget {
    int r;                // sugár
    bool fill;           // kitöltött-e a kör
public:
    Circle(const Point& p0, int r, const Color& fg = WHITE,
           const Color& bg = BLACK, bool fill = false)
        :Widget(p0, fg, bg), r(r), fill(fill){}

    void draw() const;
    bool contains(const Point& p);
    bool eventHandler(const SDL_Event& ev);
    ~Circle() {
        fg = bg;
        draw();           // letöltés
    }
};
```

Container

```
class Container : public Widget {
protected:
    std::vector<Widget*> widgets;    //< Itt tárolunk
    Point cursor;                  //< egérmozgáshoz
public:
    void add(Widget *w);
    void clear();
    bool contains(const Point& p) {
        cursor = p;                // egérmozgás követése
        return false;
    }
    bool eventHandler(const SDL_Event&);
    void draw() const;
    ~Container() { clear(); }
};
```


Bboard

```
// Hazudós, mert minden koordinátát magáénak gondol.
// Így meghívódik az eseménykezelője
class Bboard : public Container {
public:
    // A teljes képernyő a tárolóhoz tartozik.
    // Minden pontot magáénak tekint.
    bool contains(const Point& p) {
        Container::contains(p);
        return true;
    }
};
```

AppShell

```
struct AppShell : public Container {
    AppShell(SDL_Surface *scr) {
        Widget::setSurface(scr);
    }
    void mainLoop();
};

void AppShell::mainLoop() {
    SDL_Flip(scr); // megjeleníti a képet (SDL 1)
    SDL_Event ev;
    Point cursor;
    while (SDL_WaitEvent(&ev) && ev.type != SDL_QUIT) {
        bool changed = false;
        // egér követése
        if (ev.type == SDL_MOUSEMOTION) {
            cursor.x = ev.motion.x;
            cursor.y = ev.motion.y;
        }
    }
}
```

AppShell /2

```
// esemény továbbítása
size_t i = widgets.size()-1;
// megváltozhat a container tartalma a ciklus alatt
while (i >= 0 && i < widgets.size()) {
    if (widgets[i]->contains(cursor)) {
        if (widgets[i]->eventHandler(ev)) {
            changed = true;
            break;
        }
    }
    i--;
}
if (changed) {
    draw();
    SDL_Flip(scr); // SDL 1 !!!
}
}
```

Container add és clear

```
// Új widgetet ad a tárolóhoz
void Container::add(Widget *w) {
    widgets.push_back(w);
    w->draw();
}

// Törli a tárolót
void Container::clear() {
    for (size_t i = 0; i < widgets.size(); i++)
        delete widgets[i];
    widgets.clear();
}
```

Container draw és event

```
void Container::draw() const {
    for (size_t i = 0; i < widgets.size(); i++)
        widgets[i]->draw();
}

// Az objektumok sorrendje meghatározza a takarásokat.
bool Container::eventHandler(const SDL_Event& ev) {
    // megváltozhat a container tartalma a ciklus alatt
    for (size_t i = widgets.size()-1; i >= 0
        && i < widgets.size(); i--) {
        if (widgets[i]->contains(cursor)) {
            if (widgets[i]->eventHandler(ev))
                return true;
        }
    }
    return false;
}
```

Button template

```
// Jobban illeszthető a callback függvényekhez
template<class T = void (&)(const SDL_Event&, int) >
class Button : public Circle {
    T& cb;        //< visszahívandó objektum referenciája
    int val;     //< int érték
public:
    // Konstruktor elteszi a rutin címét és a paramétert
    Button(const Point& p0, int r, T& cb, int val = 0,
           const Color& fg = WHITE, const Color& bg = BLACK) :
        Circle(p0, r, fg, bg, true), cb(cb), val(val) { }
    // A felhasználói felületen megnyomták a gombot
    bool eventHandler(const SDL_Event& ev) {
        if (ev.type == SDL_MOUSEBUTTONDOWN) {
            if (ev.button.button == SDL_BUTTON_LEFT) cb(ev, val);
            return true;
        }
        return Circle::eventHandler(ev);
    }
};
```