Programozás alapjai II.
(10. ea) **C++**

STL algoritmusok

Szeberényi Imre

BME IIT

<szebi@iit.bme.hu>

MŰEGYETEM 1782

Előző óra összefoglalása

- Kivételkezelés
  - működés részletei,
  - hatása az objektumok élettartamára
  - Konstruktorban, destruktorban
- Létrehozás/megsemmisítés működésének felhasználása (auto_ptr, fájlkezelés)
- STL bevezető (tárolók)

STL tárolók összefoglalása

- A tárolók nem csak tárolják, hanem "birtokolják is az elemeket"
  - elemek létrehozása/megszüntetése
- Két fajta STL tároló van:
- Sorozat tárolók (vector, list, deque)
  - A programozó határozza meg a sorrendet:
- Asszociatív tárolók (set, multiset, map, multimap)
  - A tároló határozza meg a tárolt sorrendet
  - Az elemek egy kulccsal érhetők el.

STL tárolók összefoglalása /2

- vector<T, Alloc>
- list<T, Alloc>
- deque<T, Alloc>
- map<Key, T, Cmp, Alloc>
- set<Key, Cmp, Alloc>
- stack<T, deque>
- queue<T, deque>
- priority_queue<T, vector, Cmp>

Tárolók fontosabb műveletei

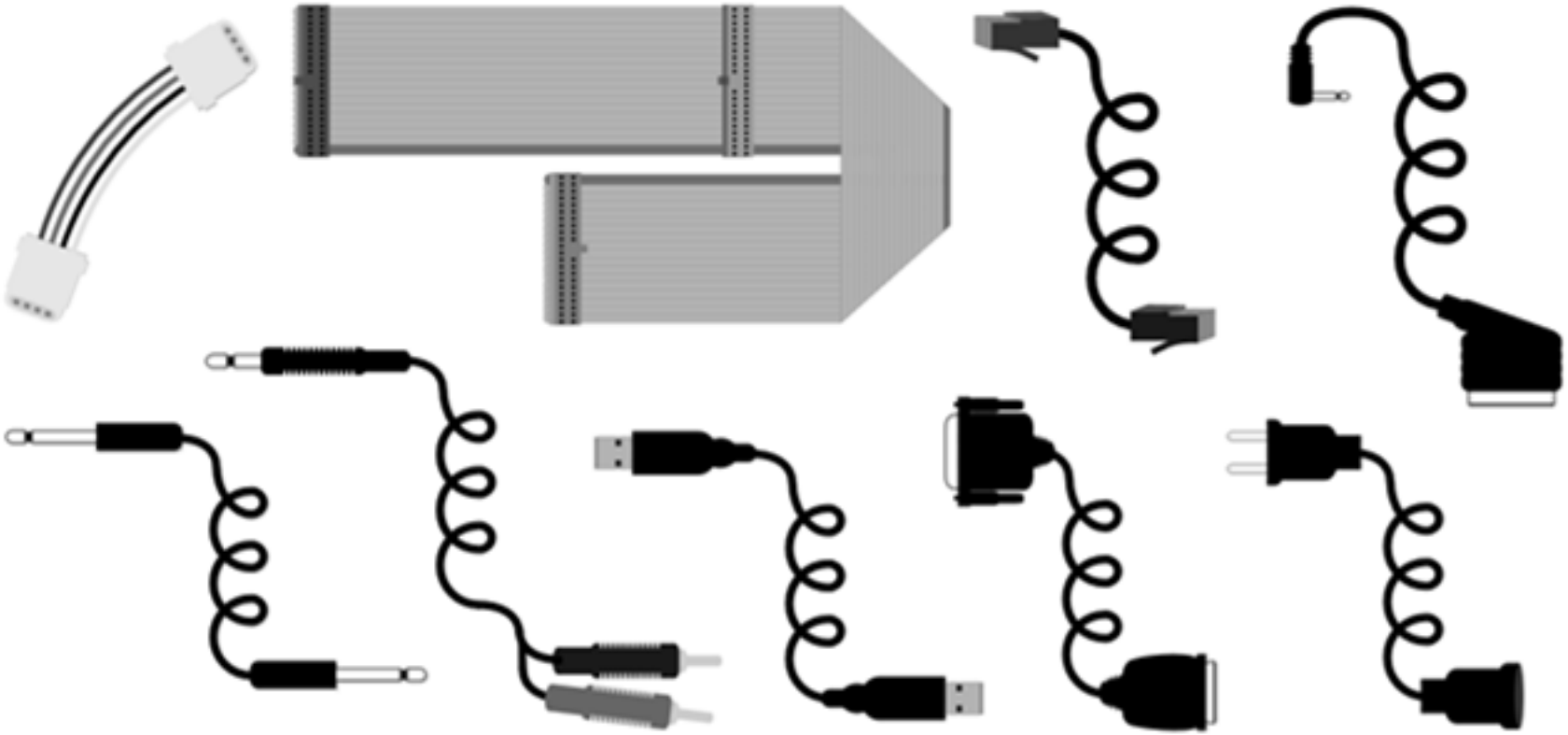
	Konstr. destr. op=	Iterátorok	size	max_size	empty	resize	front	back	op[]	at	assign	insert	erase	swap	clear	push_front	pop_front	push_back	pop_back
vector	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+			+	+
deque	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
list	+	+	+	+	+	+	+	+			+	+	+	+	+	+	+	+	+
set	+	+	+	+	+							+	+	+	+				
multiset	+	+	+	+	+							+	+	+	+				
map	+	+	+	+	+				+			+	+	+	+				
multimap	+	+	+	+	+							+	+	+	+				

Ma

- Adapter tervezési minta
- STL algoritmusok
- Korábbi példa továbbfejlesztése
  - STL tároló használata
  - STL iterátor használata
  - Egy tipikus viselkedési minta és megvalósítása
    - Observer

Adapterek

Másként szeretnénk elérni, és/vagy kicsit másként szeretnénk működtetni



std::vektor → gyak::Array

- gyak::Array (gyakorlaton készített tömb):
  - fix méretű (*maxsize*)
  - van aktuális mérete (*siz*)
  - *at()* és *operator[]*, de nyújtja a tömböt (*maxsize*-ig)
  - van iterátora (*iterator*)
  - konstruktorai a konténereknél megszokottak
  - van *operator=*
- std::vector:
  - minden van, de nem nyújtja a tömböt az indexelés és az *at()*, de dinamikusan képes növekedni (*push_back*)

Egy lehetséges megvalósítás #1

```
template <typename T, size_t maxsiz = 6>
struct Array : public std::vector<T> {
  Array(size_t n = 0, const T& value = T())
        :std::vector<T>::vector(n, value) {}

  template <class Iter>
  Array(Iter first, Iter last)
        :std::vector<T>::vector(first, last) {}

  T& operator[](size_t i) {
     if (i < maxsiz && i >= std::vector<T>::size())
       std::vector<T>::resize(i+1);
     return std::vector<T>::operator[](i);
  }
  T operator[](size_t i) const {
     return std::vector<T>::operator[](i);
  }
```

Egy lehetséges megvalósítás #2

```
  T& at(size_t i) {
     if (i >= maxsiz)
       throw std::out_of_range("Array index");
     return (*this)[i];
  }
  T at(size_t i) const {
     if (i >= maxsiz)
       throw std::out_of_range("Array index");
     return (*this)[i];
  }
};
```

std::vector minden tagfüggvénye az öröklés révén publikálva, a nem megfelelőeket módosított működéssel megvalósítottuk.
Ha nem jó, lehet privát örökléssel is, ekkor minden fv-hez kell interfész.
Szóba jöhet még a tartalmazás (delegálás) is.

Apróbb működési különbségek #1

```
template <typename T, size_t maxsiz = 6>
struct Array : public std::vector<T> {
  Array(size_t n = 0, const T& value = T())
    :std::vector<T>::vector(n, value) {}
// nagyobb lehet maxsiz-nél
// nincs maxsiz-ig lefoglalva


  Array(size_t n = 0, const T& value = T())
    :std::vector<T>::vector(std::min(n,maxsiz), value) {
    std::vector<T>::reserve(maxsiz);
  }
```

Apróbb működési különbségek #2

```
template <typename T, size_t maxsiz = 6>
  template <class Iter>
  Array(Iter first, Iter last)
        :std::vector<T>::vector(first, last) {}
// nagyobb lehet maxsiz-nél
// nincs maxsiz-ig lefoglalva


  template <class Iter>
  Array(Iter first, Iter last)
  :std::vector<T>::vector(first, last) {
     if (std::vector<T>::size() > maxsiz)
       std::vector<T>::resize(maxsiz);
     else
       std::vector<T>::reserve(maxsiz);
  }
```

Algoritmusok <algorithm>

- Nem módosító sorozatműveletek
- Sorozatmódosító műveletek
- Rendezés, rendezett sorozatok műveletei
- Halmazműveletek
- Kupacműveletek
- Mimimum, maximum
- Permutációk

Nem módosító műv.

- for_each(fisrt, last, fn)
- find(fisrt, last, val),
- find_if(fisrt, last, un_pred)
- find_end(f1, l1, f2, l2, un_pred),
- find_first_of(f1, l1, f2, l2, un_pred),
- adjacent_find(fisrt, last, bin_pred)
- count(fisrt, last)
- count_if(fisrt, last, un_pred)
- mismatch(f1, l1, f2, l2, bin_pred) // ret: pair
- equal(f1, l1, l2, bin_pred)
- search(f1, l1, f2, l2, bin_pred)
- search_n(f, l, count, val, bin_pred)

1. Példa: count_if

```
template <class InpIterator, class UnPredicate >
prtdiff_t count_if(InpIterator first, InpIterator last,
                                        UnPredicate pred ) {
  prtdiff_t ret = 0;
  while (first != last)
        if (pred(*first++)) ++ret;
  return ret;
}
```

```
int v[] = {11, 2, 3, 32, 21, 15} ;
bool IsOdd(int i) { return ((i%2)==1); }
cout << count_if (v, v+6, IsOdd); // az int* is iterator!
```

2. Példa: adjacent_find

```
template <class FwIterator, class BinPredicate >
FwIterator adjacent_find(FwIterator first,
      FwIterator last, BinPredicate pred ) {
  if (first != last) {
     FwIterator next=first;
    ++next;
     while (next != last)
         if (pred(*first++, *next++))
             return first;
  }
  return last;
}
```

3. Példa: mismatch

```
template <class Iter1, class Iter2, class BinPred>
pair<Iter1, Iter2> mismatch(Iter1  first1, Iter1 last1,
                            Iter2 first2, BinPred pred)  {
   while (first1 != last1) {
      if (!pred(*first1,*first2))
         break;
      ++first1; ++first2;
   }
   return make_pair(first1, first2);
}
```

Sorozat módosító műv.

- copy()
- copy_backward()
- swap(), iter_swap()
- swap_ranges()
- replace()
- replace_if()
- replace_copy()
- replace_copy_if()
- fill(), fill_n()
- generate()
- generate_n()
- partition()
- stable_partition()
- remove()
- remove_if()
- remove_copy()
- remove_copy_if()
- unique()
- unique_copy_if()
- reverse()
- reverse_copy()
- rotate(), rotate_copy()
- random_shuffle()
- transform()

Rendezés, rend.sor. műv., halmaz

- sort(), stable_sort(), partial_sort()
- partial_sort_copy()
- nth_element()
- lower_bound(), upper_bound()
- equal_range()
- binary_search()
- merge()
- inplace_merge()
- includes()
- set_union(), set_intersection(), set_difference()
- set_symmetric_difference()

Kupac, min, max, permut.

- make_heap()
- push_heap()
- pop_heap()
- sort_heap()
- min(), max()
- min_element(), max_element()
- lexicographical_compare()

- next_permutation()
- prev_permutation()

4. példa

```
int v[] = {10,20,30,30,20,10,10,20};
int *ip = adjacent_find (v, v+8);    // a pointer is iterator!
vector<int> iv(v, v+8);
vector<int>::iterator it;
it = adjacent_find(iv.begin(), iv.end()); // első ismétlődés
it = adjacent_find(++it, iv.end());       // második ism.


it = adjacent_find(iv.begin(), iv.end(), greater<int>());
```

predikátum

Függvényobjektumok <functional>

- unary_function, binary_function

```
template <class Arg, class Result>
  struct unary_function {
     typedef Arg argument_type;
     typedef Result result_type;
};
struct Valami : public uanry_function<int, bool> {
   ..........
}; ..........
   Valami::argument_type ........
   Valami::result_type ........
   .....
```

Predikátumok és aritm. műv.

- equal_to,
- not_equal_to,
- greater, less,
- greater_equal,
- less_equal
- logical_and,
- logical_or
- logical_not

- plus
- minus
- multiplies
- divides
- modulus
- negate

Lekötők, átalakítók, típusok

- bind2nd()
- bind1st()

- mem_fun()
- mem_fun_ref()
- prt_fun()

- not1()
- not2()

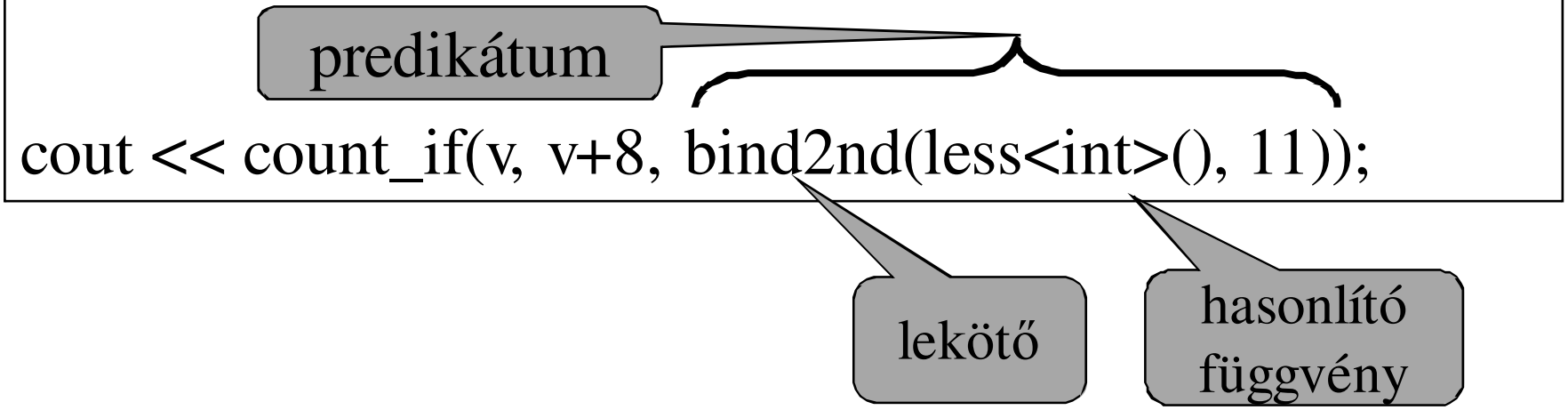
- binder1st
- binder2nd

- mem_fun1_ref_t
- mem_fun1_t
- mem_fun_ref_t
- mem_fun_t

- unary_negate
- binary_negate

5. példa

```
int v[] = {10,20,30,30,20,10,10,20};
cout << count_if(v, v+8, bind2nd(less<int>(), 11));
```



6. példa

```
bool odd(int i) { return i&1; } ....

int v[] = {10,20,15,35,92}; vector<int> iv(v, v+5);
make_heap(iv.begin(), iv.end());
print(iv);  // 92, 35, 15, 10, 20,
sort_heap(iv.begin(), iv.end());
print(iv);  // 10, 15, 20, 35, 92,

vector<int>::iterator it =
                        remove_if(iv.begin(), iv.end(), odd);
iv.erase(it, iv.end());
print(iv);  // 10, 20, 92,
```

Print template

7. példa

```
int pow(int i) { return i*i; } ....

int v[] = {1,2,5,7,10}; vector<int> iv(v, v+5);
transform(iv.begin(), iv.end(), iv.begin(), pow);
print(iv);  // 1, 4, 25, 49, 100,

iv.pop_back(); iv.pop_back();
do {

    print(iv);

} while (next_permutation(iv.begin(), iv.end()));
```

```
1, 4, 25,
1, 25, 4,
4, 1, 25,
4, 25, 1,
25, 1, 4,
25, 4, 1,
```

8. példa: Szavak gyakorisága

```
// Szavakat olvasunk, de eldobjuk a számjegyeket
bool isDigit(char ch) {  return isdigit(ch) != 0; }

map<string, int> szamlalo;
string szo;
while (cin >> szo) {
    string::iterator vege =
    remove_if(szo.begin(), szo.end(), isDigit);
    szo.erase(vege, szo.end());
    if (!szo.empty())
        szamlalo[szo] += 1;
}
```

Szavak gyakorisága /2

```
// Kiírjuk a szavakat és az előfordulási számot.
// Betesszük egy vektorba a szavakat.
// A map miatt rendezett.

vector<string> szavak;
cout << "Szavak gyakorisaga:" << endl;
for (map<string, int>::iterator it = szamlalo.begin();
                                it != szamlalo.end(); it++) {
   cout << it->first << ": " << it->second << endl;
   szavak.push_back(it->first);
}
```

Szavak gyakorisága /3

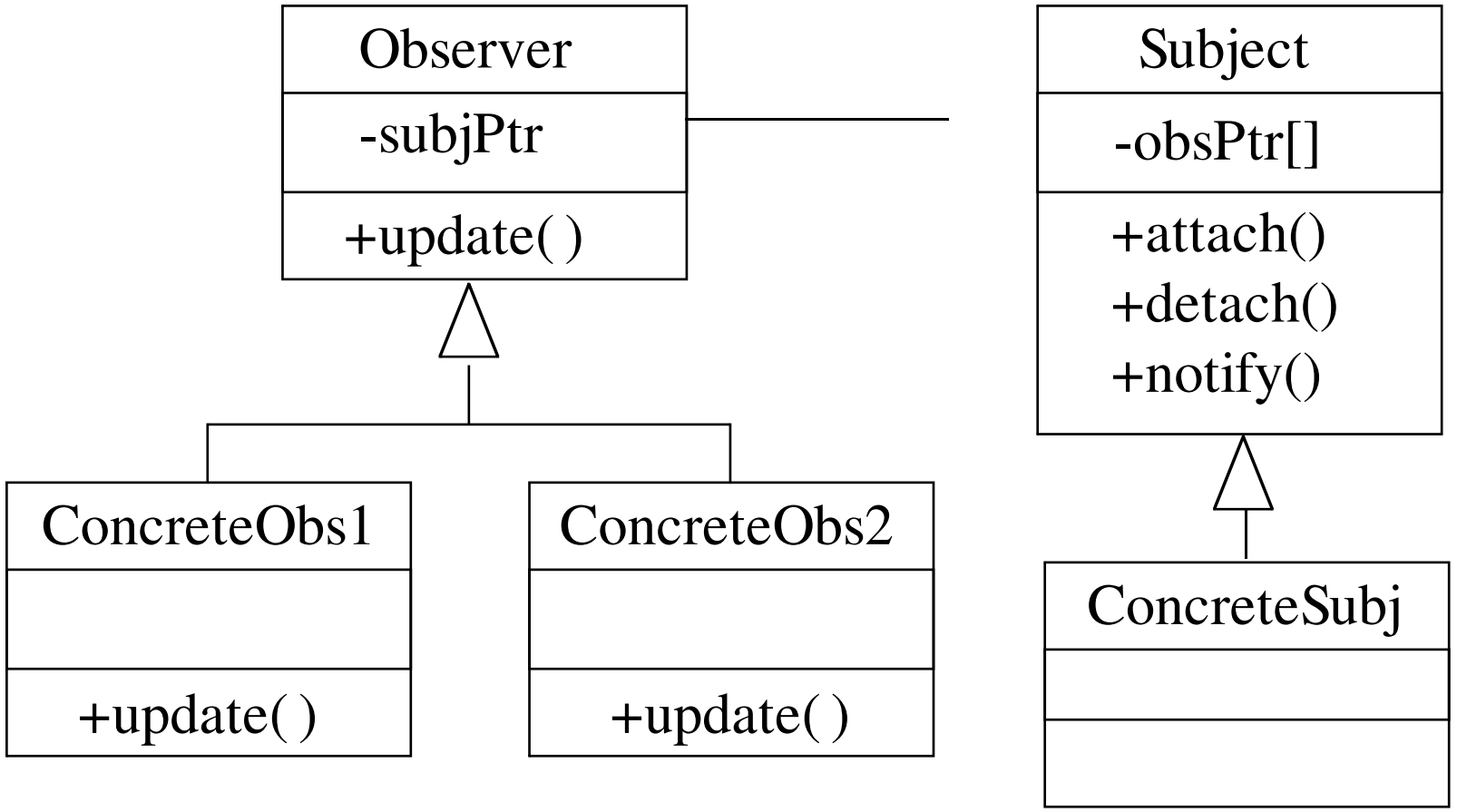
```
// Kiírjuk a szavakat a vektorból.
// Fordítva is lerendezzük.

cout << "Szavak rendezve:" << endl;
ostream_iterator<string> out_it(cout, ",");
copy(szavak.begin(), szavak.end(), out_it);

cout << endl << "Szavak forditva:" << endl;
sort(szavak.begin(), szavak.end(), greater<string>());

copy(szavak.begin(), szavak.end(), out_it);
```

Cápali és Cápeti

Feladat

- Egészítsük ki a korábbi modellünket:
  - Az állatvédők tudni akarják, hogy mekkora utat tesz meg élete során Cápeti, a cápa.
  - A tengerbiológusok tudni akarják, hogy hányszor szaporodik Cápeti.
  - A dokumentum film készül Cápeti útjáról.
- Kérdések:
  - Tegyünk 3 jeladót Cápeti nyakába?
  - 1 jeladó jelezzen mindenkinek?

Observer terv. minta



Subject osztály

```
class Subject {
   set<Observer*> obs;  // observerek pointere
public:
   void attach(Observer* os);
   void detach(Observer* os);
   void notify(int reason);
   virtual ~Subject();
};
```

Observer osztály

```
class Observer {
   Subject *subj;          // megfigyelt objektum
 public:
   Observer(Subject* subj);
   virtual void update(Subject* subj,
                                    int reason);
   virtual ~Observer();
};
```

Subject tagfüggvényei /1

```
void Subject::attach(Observer *o) {
   obs.insert(o);
}
void Subject::detach(Observer *o) {
   obs.erase(obs.find(o));
}
Subject::~Subject() {
   notify(0);                  // jelzi, hogy megszűnt
}
```

Subject tagfüggvényei /2

```
// minden figyelőt értesít a változásról
void Subject::notify(int reason) {
    for (std::set<Observer*>::iterator it =
                obs.begin(); it != obs.end(); it++ )
        (*it)->update(this, reason);
}
```

Observer tagfüggvényei

```
Observer::Observer(Subject *subj) :subj(subj){
   subj->attach(this);
}
void Observer::update(Subject* subj, int reason) {
   if (reason == 0)
      this->subj = 0;          // megszűnt: nem figyeli
}
Observer::~Observer() {
   if (subj != 0)              // van még kit figyelni ?
      subj->detach(this);
}
```

Figyelt cápa

```
class FigyeltCapa :public Capa,
                                public Subject {
   Koord lastPos;
public:
   Koord getpos() const { return lastPos; }
   void lep(Koord pos, Ocean& oc, int it);
};
```

Cápa figyelő

```
class CapaFigyelo : public Observer {
.....
public:
   CapaFigyelo(FigyeltCapa *fc);
    int getkor() const;
    int getehes() const;
    void update(Subject *subj, int oka);
    void ut(std::ostream& os);
};
```

Figyelés indítása

```
FigyeltCapa *capeti = new FigyeltCapa;
CapaFigyelo mester(capeti);
CapaFigyelo filmes(capeti);
CapaFigyelo biologus(capeti);
....
```