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

Szeberényi Imre, Somogyi Péter
BME IIT
<szebi@iit.bme.hu>

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 destruktorainak hívása
  - virtuális függvénymutatók visszaállítása
  - virtuális alaposztály mutatóinak visszaállítása
- Hívja a közvetlen, nem virtuális alaposztályok destruktorait.
- Öröklési lánc végén hívja a virtuális alaposztályok destruktorait.

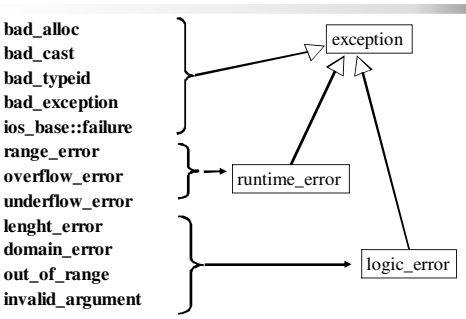
https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_09 → ctor_dtor
https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_10 → ctor_dtor2

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 kivételek (stdexcept)



Szabványos kivételek/2

```
class exception {
...
public:
   exception() throw();
   exception(const exception&) throw();
   execption& 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.

referencia

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


Szabványos tárolók

- vector
- list
- deque

} Sorozattárolók

- stack
- queue
- priority_queue

} Adapterek

- map
- multimap
- set
- multiset

} Asszociatív tárolók

- string
- array
- valarray
- bitset

} Majdnem 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)

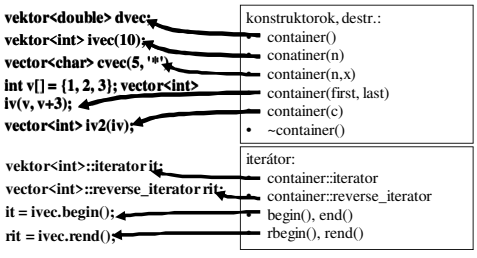
Általános, minden tárolóra érvényes
- létrehozás/megszüntetés: konstruktorok/destruktor
- értékadás: operator=, assign()
- iterátor példányosítás: begin(), end(), rbegin(), reend()
- méret lekérdezés: size(), max_size(), capacity() empty()
- módosítók: insert(), erase(), clear(), swap()

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



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);  // nem kisebb
cout << *it;                    // 4
it = set1.upper_bound(4);       // nagyobb
cout << *it;                    // 10
std::pair<myit, myit> rit;
rit = set1.equal_range(4);      // [ lower_bound(4); upper_bound(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)

Nincs:
- push_front, pop_front
- asszociatív op.

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

```
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>

Kétvégű 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>

Elrejti 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>

Elrejti 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,
```


STL tárolók összefoglalása

- A tárolók nem csak tárolják, hanem "birtokolják is az elemeket"
  - elemek létrehozása/megszüntetése/másolása
- 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=	Ite-ri-to-rok	size	capacity	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	+	+	+		+	+							+	+	+	+				

Algoritmusok <algorithm>

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

Nem módosító műv.

- for_each(first, last, fn)
- find(first, last, val),
- find_if(first, last, un_pred)
- find_end(f1, l1, f2, l2, un_pred),
- find_first_of(f1, l1, f2, l2, un_pred),
- adjacent_find(first, last, bin_pred)
- count(first, last)
- count_if(first, 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: 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);
}
```


3. 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;
}
```


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, ha hiányzik akkor ==

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()

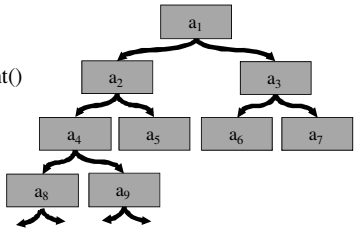
Kupac, min, max, permut.

- make_heap()
- push_heap()
- pop_heap()
- sort_heap()

- min(), max()
- min_element(), max_element()
- lexicographical_compare()

- next_permutation()
- prev_permutation()

$a_i >= a_{2i}$ ha i < n/2
$a_i >= a_{2i+1}$ ha i < n/2



5. 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

6. 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,
```


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 unary_function<int, bool> {
   .........
}; .........
   Valami::argument_type ........
   Valami::result_type ........
   .....
```

✞ C++17-től megszűnt helyette function template és lambda

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 ✞

- **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**

✞ C++17-től megszűnt helyette bind és lambda

7. példa

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

predikátum

lekötő

hasonlító függvény

✞ C++17-től megszűnt helyette bind és lambda

```
count_if(v, v+8, bind(less<int>(), std::placeholders::_1, 11));
```


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);
```

https://git.ik.bme.hu/Prog2/eloadas_peldak/ea_11 → szostatisztika

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.

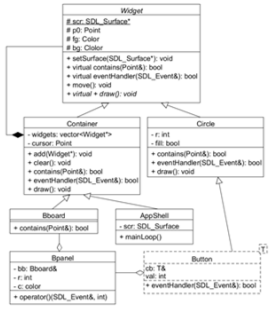
Események: Egérmozgatás, kattintás, húzás (drag)

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

Szereplők

- grafikus primitívek:
  - Widget, Circle, Button, ActiveBg
- összetettet 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 értékel 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();         // letörlés
  }
};
```


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);
  }
};
```