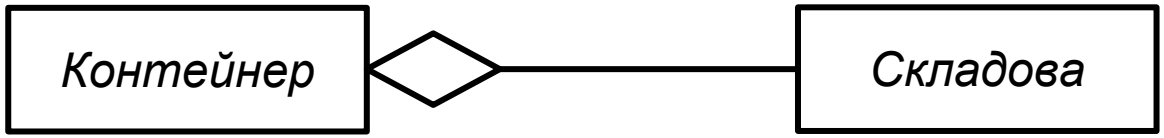
Агрегація та композиція

Повторне використання коду в складних об’єктах

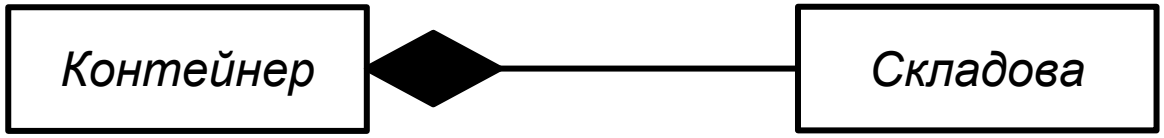
Відношення **агрегації** - більш слабке відношення типу “частина-ціле” між класом-складовою та класом-контейнером,

при якому між об’єктами класів **не існує зв’язку за часом існування**: складова може існувати після знищення контейнера



Відношення **композиції** - більш сильне відношення типу “частина-ціле” між класом-складовою та класом-контейнером,

при якому об’єкти класів **зв’язані за часом існування**: складова знищується разом з контейнером



Контейнер std::vector

Вектор є аналогом **динамічного масиву**, що автоматично виділяє пам'ять під нові елементи. Виділення додаткової пам'яті відбувається за вказівкою **reserve (N)** або автоматично (тоді, як правило, розмір вектору подвоюється).

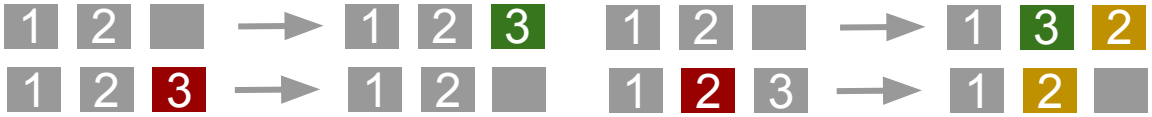
Елементи вектора розміщуються у пам'яті **послідовно без розривів:**

size () = 2, **capacity ()** = 4

- швидкий доступ до **довільного** елементу **v[i], *(v_ptr + i)**

- При нестачі пам'яті виділяється нова область більшого розміру і всі елементи **копіюються** (**copy c-tor**) до неї [**O(n)**]

- Лише вставка у **кінець** вектору **std::vector::push_back ()** та вилучення **std::vector::pop_back ()** з кінця вектору є ефективними [**O(1)**], інакше елементи **копіюються** на сусідні місця [**O(n)**]



Використовуйте, якщо сумніваєтесь щодо вибору контейнеру
та **замість звичайних динамічних масивів**

Контейнер std::list

Двузв'язний список розміщує елементи у пам'яті **довільно:**

Доступ до **довільного** елементу перебором
через **ітератор послідовного доступу** [***O(n)***]

Розмір кожного елемента включає пам'ять
для **вказівників** "вперед" та "назад"

Вставка **insert ()** або видалення **delete ()** елементів будь-де за [***O(1)***]
Жодних накладних копіювань.

Вказівники на елементи залишаються **дійсними** при будь-яких операціях над усіма елементами списку, окрім тих, на які вони вказують

Контейнери послідовностей

Заголовочний файл		<array>	<vector>	<deque>	<forward_list>	<list>
Контейнер		**array**	**vector**	**deque**	**forward_list**	**list**
Ітератори	**begin**	begin	begin	begin	begin before_begin	begin
	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend	rend		rend
Ємність	**size**	size	size	size		size
	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
	reserve		reserve			

Контейнери послідовностей

Заголовочний файл		<array>	<vector>	<deque>	<forward_list>	<list>
Контейнер		**array**	**vector**	**deque**	**forward_list**	**list**
Доступ до елементів	**front**	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
	at	at	at	at		
Модифікат.	**swap**	swap	swap	swap	swap	swap
	assign		assign	assign	assign	assign
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	clear		clear	clear	clear	clear
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back

Приклади коду

```
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  for (int i=1; i<=5; i++) myvector.push_back(i);

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)
  {
    std::cout << ' ' << *it;
  }
}
```

```
#include <iostream>
#include <list>

int main ()
{
  int myints[] = {1,2,3,4,5};
  std::list<int> mylist (myints,myints+5);

  std::cout << "mylist contains:";
  for (std::list<int>::iterator it=mylist.begin(); it != mylist.end(); ++it)
  {
    std::cout << ' ' << *it;
  }
  for (auto it=mylist.begin(); it != mylist.end(); ++it)
  {
    std::cout << ' ' << *it;
  }
}
```

Приклади коду

```
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector (3,100);
  std::vector<int>::iterator it;

  it = myvector.begin();
  it = myvector.insert ( it , 200 );

  myvector.insert (it,2,300);

  // "it" вже не дійсний!
  it = myvector.begin();

  std::vector<int> anothervector (2,400);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());

  int myarray [] = { 501,502,503 };
  myvector.insert (myvector.begin(), myarray, myarray+3);

  std::cout << "myvector contains:";
  for (int i=0; i<myvector.size(); ++i)
    std::cout << ' ' << myvector[i];
}
```

Приклади коду

```
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;

  // set some initial content:
  for (int i=1;i<10;i++) myvector.push_back(i);

  myvector.resize(5);
  myvector.resize(8,100);
  myvector.resize(12);

  std::cout << "myvector contains:";
  for (int i=0;i<myvector.size();i++)
    std::cout << ' ' << myvector[i];
}
```

```
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector (100);
  std::cout << "1. capacity of myvector: " << myvector.capacity() << '\n';
  myvector.resize(10);
  std::cout << "2. capacity of myvector: " << myvector.capacity() << '\n';
  myvector.shrink_to_fit();
  std::cout << "3. capacity of myvector: " << myvector.capacity() << '\n';
}
```

Приклади коду

```
#include <iostream>
#include <deque>
#include <vector>

int main ()
{
  std::deque<int> mydeque;

  // set some initial values:
  for (int i=1; i<6; i++) mydeque.push_back(i);

  std::deque<int>::iterator it = mydeque.begin();
  ++it;

  it = mydeque.insert (it,10);

  mydeque.insert (it,2,20);

  it = mydeque.begin()+2;

  std::vector<int> myvector (2,30);
  mydeque.insert (it,myvector.begin(),myvector.end());

  std::cout << "mydeque contains:";
  for (it=mydeque.begin(); it!=mydeque.end(); ++it)
    std::cout << ' ' << *it;

}
```

Приклади коду

```
#include <iostream>
#include <deque>

int main ()
{
  std::deque<int> mydeque (10);
  std::deque<int>::size_type sz = mydeque.size();

  for (unsigned i=0; i<sz; i++) mydeque[i]=i;

   for (unsigned i=0; i<sz/2; i++)
  {
    int temp;
    temp = mydeque[sz-1-i];
    mydeque[sz-1-i]=mydeque[i];
    mydeque[i]=temp;
  }

  std::cout << "mydeque contains:";
  for (unsigned i=0; i<sz; i++)
    std::cout << ' ' << mydeque[i];

  return 0;
}
```

Приклади коду

```
#include <iostream>
#include <algorithm>
#include <vector>

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

  std::sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26 80 53 33

  std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)

  std::sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;

}
```

Адаптери: стек

```
#include <iostream>
#include <stack>
#include <vector>
#include <deque>

int main ()
{
  std::deque<int> mydeque (3,100);
  std::vector<int> myvector (2,200);

  std::stack<int> first;
  std::stack<int> second (mydeque);

  std::stack<int,std::vector<int> > third;
  std::stack<int,std::vector<int> > fourth (myvector);

  std::cout << "size of first: " << first.size() << '\n';
  std::cout << "size of second: " << second.size() << '\n';
  std::cout << "size of third: " << third.size() << '\n';
  std::cout << "size of fourth: " << fourth.size() << '\n';

  return 0;
}
```

Адаптери: черга

```
include <iostream>
#include <queue>

int main ()
{
  std::queue<int> foo,bar;
  foo.push (10); foo.push(20); foo.push(30);
  bar.push (111); bar.push(222);

  swap(foo,bar);

  std::cout << "size of foo: " << foo.size() << '\n';
  std::cout << "size of bar: " << bar.size() << '\n';

  return 0;
}
```

PIMPL idiom: pointer to implementation

Якщо потрібно приховати усі деталі реалізації. Int.cpp може бути попередньо скомпільований (постачатися у вигляді бібліотеки).

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

class Int
{
public:
     Int();
     ~Int();
     void print();
private:
     class Imp;
     Imp* imp_;
};



int main()
{
     Int a;
     a.print();
}
```

Int.cpp

```
class Int::Imp
{
public:
     void print()
     {
          cout << "Imp.print()" << endl;
     }
};

Int::Int():imp_(new Imp){}

Int::~Int()
{
   delete imp_;
}

void Int::print()
{
   imp_->print();
}
```

Шаблони

```
template <typename T>
class Unsigned
{
public:

     Unsigned(T number = 0)
     : number(number < 0 ? -number : number)
     {}

     operator T() const { return number; }

     Unsigned & operator *= (Unsigned multiplier)
     {
         number *= multiplier.number;
         return *this;
     }

private:

     Numeric number;
};
```

Шаблони - механізм узагальненого програмування. Дозволяють реалізувати узагальнені (не прив'язані до конкретних параметрів, наприклад - типів даних) алгоритми.

C++ допускає створення шаблонів функцій (включаючи методи) та шаблони класів.

```
double number = -5.5;
Unsigned<double> un = number;
std::clog << un << std::endl;
```

```
template <class T>
Unsigned<T> operator* (Unsigned<T> left, Unsigned<T> right)
{
     left *= right;
     return left;
}
```

Шаблони

```
template< typename T >
void func1( T a[], int n, T x, T y)
{
  for (int i = 0; i < n; i++)
  {
     if (a[i] == x)
         a[i] = y;
  }
}
```

```
template< typename T >
void func2( T a[], int n )
{
  for (int i = 0; i < n / 2; i++)
  {
    T t;
    t = a[n - 1 - i];
    a[n - 1 - i] = a[i];
    a[i] = t;
  }
}
```

```
template< typename T >
void func3( T a[], int n )
{
  T t;
  for (int i = 0; i < n - 1; i++)
    for (int j = n - 1; j > i; j--)
      if (a[j] < a[j-1])
      {
        t = a[j];
        a[j] = a[j-1];
        a[j-1] = t;
      }
}
```

```
template< typename T >
void func4( T a[], int n, T x)
{
  for (int i = 0; i < n; i++)
  {
     if (a[i] == x)
     {
        for (int j = i; j < n - 1; j++)
        {
            a[j] = a[j + 1];
        }
        n--;
     }
    }
}
```

```
template< typename T >
void func5( T a[], int n, int m)
{
  int i = 0;
  int j = m;
  while (i != j)
  {
     T t = a[i];
     a[i++] = a[j];
     a[j++] = t;
     if (j == n) j = m;
     else if (i == m) m = j;
  }
}
```

Готові шаблони зі стандартної бібліотеки

```
template <class ForwardIterator, class T>
  void replace (ForwardIterator first,
                ForwardIterator last,
                const T& old_value,
                const T& new_value)
{
  while (first!=last) {
    if (*first == old_value)
*first=new_value;
    ++first;
  }
}
```

```
template <class BidirectionalIterator>
  void reverse (BidirectionalIterator first,
                BidirectionalIterator last)
{
  while ((first!=last)&&(first!=--last))
  {
    std::iter_swap (first,last);
    ++first;
  }
}
```

std::**sort**,
std::stable_sort

```
template <class ForwardIterator, class T>
  ForwardIterator remove (ForwardIterator firs
                          ForwardIterator last
                          const T& val)
{
  ForwardIterator result = first;
  while (first!=last) {
    if (!(*first == val)) {
      *result = *first;
      ++result;
    }
    ++first;
  }
  return result;
}
```

```
template <class ForwardIterator>
  void rotate (ForwardIterator first,
               ForwardIterator middle,
               ForwardIterator last)
{
  ForwardIterator next = middle;
  while (first!=next)
  {
    swap (*first++,*next++);
    if (next==last) next=middle;
    else if (first==middle) middle=next;
  }
}
```

Асоціативні контейнери. Map - словник “ключ-значення”

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

struct Person
{
     int age;
     string name;
};

int main ()
{
     map < string, Person* > c1;
     map < int, Person* > c2;
     Person p1; p1.age = 20; p1.name = "Bill";
     Person p2; p2.age = 35; p2.name = "Andrew";
     Person p3; p3.age = 15; p3.name = "Max";
     c1[p1.name] = &p1; c1[p2.name] = &p2; c1[p3.name] = &p3;
     c2.insert(pair<int, Person*>(p1.age, &p1));
     c2.insert(pair<int, Person*>(p2.age, &p2));
     c2.insert(pair<int, Person*>(p3.age, &p3));
     for (map < string, Person* >::iterator i = c1.begin();
     i != c1.end(); ++i)
     {
          cout << i->second->name << endl;
     }
     for (map < int, Person* >::iterator i = c2.begin();
     i != c2.end(); ++i)
     {
          cout << i->second->name << endl;
     }
}
```

Асоціативні контейнери. Map & Set

Тест перевірки створення та видалення об’єктів у контейнерах.

```
#include <iostream>
#include <string>
#include <set>
#include <map>
using namespace std;

class Element
{
     int d;
     int i;
     static int c;
public:
     int getData() const {return d;}
     int getIdx() const {return i;}

     Element():d(0),i(++c){cout << "+e" << i << "()" << endl;}
     Element(int a):d(a),i(++c){cout << "+e" << i << "(" << d << ")" << endl;}
     Element(const Element & e):d(e.d),i(++c)
     {
          cout << "+e" << i << "(e"<< e.i<<"{" << e.d << "})" << endl;
     }
     ~Element() {cout << "~e" << i << "{" << d << "}" << endl;}
     Element & operator=(const Element & e)
     {
          cout << "e"<<i<<"{" << d << "} = e"<<e.i<<"{" << e.d << "}" << endl;
          d = e.d;
          return *this;
     }
};
```

Асоціативні контейнери. Map & Set

```
int Element::c = 0;

struct ElementCompare {
  bool operator() (const Element& l, const Element& r) const
  {return l.getData() < r.getData();}
} comparator;

ostream & operator<< (ostream& out, const Element & e)
{
     out << "e" << e.getIdx() << "{" << e.getData() << "}";
     return out;
}

template< typename T>
void print(const T & c, ostream & out = cout)
{
     out << "size " << c.size() << " ";
     for (typename T::const_iterator i = c.begin(); i != c.end(); ++i)
     {
          out << *i << " ";
     }
     out << endl;
}

template< typename T>
void printm(const T & c, ostream & out = cout)
{
     out << "size " << c.size() << " ";
     for (typename T::const_iterator i = c.begin(); i != c.end(); ++i)
     {
          out << "{" << i->first << " = " << i->second << "} ";
     }
     out << endl;
}
```

Асоціативні контейнери. Map & Set

```
int main()
{
 cout << "\nset\n";
 set< Element, ElementCompare > s;
 s.insert(Element(2));
 s.insert(Element(1));
 print< set< Element, ElementCompare > >(s);
 set< Element, ElementCompare >::iterator i = s.begin();
 const_cast<Element &>(*i) = Element(3);
 print< set< Element, ElementCompare > >(s);

 cout << "\nmap\n";
 typedef map<int, Element> Map;
 Map m;
 m.insert(Map::value_type(5, Element(50)));
 m.insert(Map::value_type(1, Element(10)));
 printm(m);
 Map::iterator j = m.find(1);
 if (j != m.end())
 {
     Element tmp = j->second;
     m.erase(j);
     m.insert(make_pair(10, tmp));
 }
 m[10] = Element(100);
 m[5] = Element(500);
 printm(m);
```

```
set
+e1(2)
+e2(e1{2})
~e1{2}
+e3(1)
+e4(e3{1})
~e3{1}
size 2 e4{1} e2{2}
+e5(3)
e4{1} = e5{3}
~e5{3}
size 2 e4{3} e2{2}
```

```
map
+e6(50)
+e7(e6{50})
+e8(e7{50})
~e7{50}
~e6{50}
+e9(10)
+e10(e9{10})
+e11(e10{10})
~e10{10}
~e9{10}
size 2 {1 = e11{10}} {5 = e8{50}}
+e12(e11{10})
~e11{10}
+e13(e12{10})
+e14(e13{10})
+e15(e14{10})
+e16(e15{10})
~e15{10}
~e14{10}
~e13{10}
~e12{10}
+e17(100)
e16{10} = e17{100}
~e17{100}
+e18(500)
e8{50} = e18{500}
~e18{500}
size 2 {5 = e8{500}} {10 = e16{100}}
~e16{100}
~e8{500}
~e2{2}
~e4{3}
```

Асоціативні контейнери. UnorderedMap

Unordered-версії контейнерів (C++11) не підтримують відсортований порядок об’єктів (за допомогою бінарних дерев), але дозволяють швидкий пошук елемента по ключу (за допомогою хеш-функцій)

```
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;

template < typename K, typename V >
void print (const unordered_map< K, V >& m)
{
  cout << "size: " << m.size() << " / " << m.max_size()
     << " buckets: " << m.bucket_count() << " / " << m.max_bucket_count()
     << " load: " << m.load_factor() << " / " << m.max_load_factor() << endl;
}

template < typename K, typename V >
void print_all (const unordered_map< K, V >& m)
{
  print(m);

  for (unsigned i = 0; i < m.bucket_count(); ++i)
  {
     int bsz = m.bucket_size(i);
     if (bsz)
     {
     cout << "bucket " << i << " has " << bsz << " elements.\n";
     }
  }

  for (auto & i : m)
  {
     cout << "[" << i.first << " -> " << m.hash_function() (i.first)
          << " mod " << m.bucket_count() << " = " << m.hash_function() (i.first) % m.
bucket_count()
          << " (" << m.bucket(i.first) << " bucket)] = " << i.second << endl;
  }
}
```

Асоціативні контейнери. UnorderedMap

```
int main()
{
  unordered_map<string,string> m;

  m["01"] = "a";
  m["02"] = "b";
  m["03"] = "c";
  m["04"] = "d";
  m["05"] = "e";
  m["06"] = "f";

  print_all(m);

  unordered_map<int,int> m2;
  for (int i = 0; i < 10; ++i)
  {
     for (int j = 0; j < 10; ++j)
     {
          m2[i * 10 + j] = 10 * (i * 10 + j);
     }
     print(m2);
  }
}
```

```
g++ -std=c++11 uno.cpp

bucket 4 has 1 elements.
bucket 13 has 1 elements.
bucket 19 has 1 elements.
bucket 20 has 1 elements.
bucket 22 has 2 elements.
[05 -> 4223885067 mod 23 = 19 (19 bucket)] = e
[06 -> 3937468669 mod 23 = 22 (22 bucket)] = f
[04 -> 305468864 mod 23 = 22 (22 bucket)] = d
[03 -> 1198751227 mod 23 = 13 (13 bucket)] = c
[02 -> 2411615945 mod 23 = 4 (4 bucket)] = b
[01 -> 3451288710 mod 23 = 20 (20 bucket)] = a
size: 10 / 357913941 buckets: 23 / 357913941 load: 0.434783 / 1
size: 20 / 357913941 buckets: 23 / 357913941 load: 0.869565 / 1
size: 30 / 357913941 buckets: 53 / 357913941 load: 0.566038 / 1
size: 40 / 357913941 buckets: 53 / 357913941 load: 0.754717 / 1
size: 50 / 357913941 buckets: 53 / 357913941 load: 0.943396 / 1
size: 60 / 357913941 buckets: 109 / 357913941 load: 0.550459 / 1
size: 70 / 357913941 buckets: 109 / 357913941 load: 0.642202 / 1
size: 80 / 357913941 buckets: 109 / 357913941 load: 0.733945 / 1
size: 90 / 357913941 buckets: 109 / 357913941 load: 0.825688 / 1
size: 100 / 357913941 buckets: 109 / 357913941 load: 0.917431 / 1
```

Exceptions - обробка виключних ситуацій

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

int main () {
  try
  {
    throw 20;
  }
  catch (int e)
  {
    cout << "An exception occurred. Exception Nr. " << e << '\n';
  }
  return 0;
}
```

Виключення в конструкторі дозволяють сповістити про помилки під час створення об’єктів.

Але в той же час виключення
НЕ ПОВИННІ ЗАЛИШАТИ МЕЖІ деструктора!

```
try {
  // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

```
try {
  try {
      // code here
  }
  catch (int n) {
      throw;
  }
}
catch (...) {
  cout << "Exception occurred";
}
```

Стандартні виключення

```
#include <iostream>
#include <exception>
using namespace std;

class myexception: public exception
{
  virtual const char* what() const throw()
  {
    return "My exception happened";
  }
} myex;

int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << '\n';
  }
  return 0;
}
```

exception	description
logic_error	error related to the internal logic of the program
runtime_error	error detected during runtime
bad_alloc	thrown by new on allocation failure
bad_cast	thrown by dynamic_cast when it fails in a dynamic cast
bad_exception	thrown by certain dynamic exception specifiers
bad_typeid	thrown by typeid
bad_function_call	thrown by empty function objects
bad_weak_ptr	thrown by shared_ptr when passed a bad weak_ptr