ООП 2021
Лекция 14

Предикаты. Функторы. Алгоритмы STL. <algorithm>

oopCpp@yandex.ru

Алгоритмы стандартной библиотеки

Немодифицирующие операции над последовательностями:

all_of
any_of
none_of // Проверяют, является ли предикат верным (true) для всех (all_of), хотя бы одного из (any_of) или ни одного (none_of) из элементов в диапазоне
for_each // Применяет функцию к диапазону элементов
count
count_if // Возвращает количество элементов, удовлетворяющих определенным критериям
find
find_if
find_if_not // Находят первый элемент, удовлетворяющий определенным критериям
find_end // Ищет последнее вхождение подпоследовательности элементов в диапазон
find_first_of // Ищет в множестве элементов первое вхождение любого элемента другого множества

adjacent_find // Ищет в диапазоне два одинаковых смежных элемента

mismatch // Находит первую позицию, в которой два диапазона отличаются

equal // Определяет, одинаковы ли два множества элементов

lexicographical_compare // Возвращает истину, если один диапазон
// лексикографически меньше, чем другой

is_permutation // определяет, является ли последовательность
// перестановкой другой последовательности

search // Ищет первое вхождение последовательности элементов в диапазон

search_n // Ищет в диапазоне первую последовательность n
// одинаковых элементов, каждый из которых равен
// заданному значению

Модифицирующие операции над последовательностями:

copy
copy_if // Копирует ряд элементов
copy_n // Копирует ряд элементов в новое место
copy_backward // Копирует диапазон элементов в обратном порядке
move // перемещает диапазон элементов в новое место
move_backward // перемещает диапазон элементов в новое место в
// обратном порядке
fill // присваивает определенное значение набору элементов
fill_n // присваивает значение заданному числу элементов
transform // применяет функцию к различным элементам
generate // сохраняет результат функции в диапазоне
generate_n // сохраняет результат N раз примененной функции

remove
remove_if // удаляет элементы, удовлетворяющие определенным критериям

remove_copy
remove_copy_if // Копирует диапазон элементов опуская те, которые
// удовлетворяют определенным критериям

replace
replace_if // заменяет все значения, удовлетворяющие определенным
// критериям с другим значением
swap // обмен значения двух объектов
swap_ranges // обмен элементов в двух диапазонах
iter_swap // обмен элементов, на которые указывают итераторы
reverse // изменяет порядок элементов в диапазоне на обратный
reverse_copy // создает копию диапазон, который меняется на
// противоположную
rotate // вращает (сдвигает) последовательность элементов
// циклически до заданного элемента
rotate_copy // копирует и сдвигает в элементы диапазона
random_shuffle // перемешивает элементы на заданном диапазоне
// случайным образом
unique // удаляет все последовательные эквивалентные элементы
// кроме первого
unique_copy // создает копию некоторого диапазона элементов,
// который не содержит последовательных дубликатов

Операции разделения:

is_partitioned // определяет, разделен ли диапазон данным предикатом

partition // делит диапазон элементов на две группы
// т.е. образует два раздела в заданном диапазоне, размещая
// элементы, удовлетворяющие заданному условию, перед
// теми, которые этому условию не соответствуют.

partition_copy // копирует диапазон, разделяющий элементы на две группы

stable_partition // делит диапазон на две группы, сохраняя относительный
// порядок элементов

partition_point // находит точку разделения разделенного диапазона

Операции, относящиеся к упорядочиванию:

is_sorted // проверяет, является ли диапазон отсортированным в порядке
// возрастания
is_sorted_until // находит первый несортированный элемент в диапазоне
sort // сортирует диапазон в порядке возрастания
partial_sort // сортирует первые N элементов в диапазоне
partial_sort_copy // копирует и частично сортирует диапазон элементов
stable_sort // сортирует диапазон элементов при сохранении порядка
// между равными элементами
nth_element // помещает n-й элемент в позицию, которую он занимал бы
// после сортировки всего диапазона

Операции двоичного поиска (на отсортированных диапазонах) :

lower_bound // находит первый элемент диапазона больший чем заданное
// число или равный ему

upper_bound // находит первый элемент диапазона больший, чем
// заданное число

binary_search // определяет, находится ли элемент в некотором диапазоне

equal_range // возвращает набор элементов для конкретного ключа

Операции над множествами (на отсортированных диапазонах) :

merge // слияние двух отсортированных диапазонов

inplace_merge // слияние двух отсортированных диапазонов на месте

includes // возвращает истину, если один набор является
// подмножеством другого

set_difference // вычисляет разницу между двумя наборами

set_intersection // вычисляет пересечение двух множеств

set_symmetric_difference // вычисляет симметрическая разность между
// двумя наборами

set_union // объединяет два множества

Операции над пирамидой (кучей) :

is_heap // проверяет является ли данный диапазон пирамидой

is_heap_until // находит наибольший поддиапазон, который является кучей

make_heap // создает пирамиду из ряда элементов

push_heap // добавляет элемент в пирамиду

pop_heap // удаляет наибольший элемент из пирамиды

sort_heap // Сортировка элементов пирамиды

Некоторые алгоритмы (работа с числами) находятся в <numeric>:

Этот заголовочный файл содержит набор алгоритмов для выполнения определенных операций над последовательностями числовых значений.

Благодаря своей гибкости они также могут быть адаптированы для других видов последовательностей.

```
iota                  //  заполняет диапазон, последовательностью значений,
                      //     начиная с заданного стартового значения
accumulate            //     суммирует диапазон элементов
inner_product         //   вычисляет скалярное произведение двух диапазонов
adjacent_difference     //  вычисляет разницу между соседними элементами в
                          // диапазоне
partial_sum               //    вычисляет частичную сумму ряда элементов
```

Операции минимума/максимума:

max // Возвращает наибольший из двух аргументов
max_element // Возвращает наибольший элемент в диапазоне
min // Возвращает меньший из двух элементов
min_element // Возвращает наименьший элемент в диапазоне
minmax // Возвращает большее и меньшее из двух элементов
minmax_element // возвращает наименьший и наибольший элемент в диапазоне

Алгоритмы из библиотеки C :

<cstdlib>

qsort // Сортирует диапазон элементов любого типа

bsearch // Ищет в массиве элемент любого типа

Предикат. Функция-предикат и функтор

Предикат, это нечто функциональное, возвращающее тип bool. Есть две возможности организации такой функции: собственно функция-предикат и функтор (объект-функция).

Однако, объекты-функции (функторы) гораздо предпочтительнее. Причина проста - объекты функций обеспечивают **более эффективный** код.

В книге Скотта Мейерса на этот случай приводится пример с функцией std::sort - использование объектов функций всегда работает быстрее, выигрыш в скорости может составлять от 50% до 160%.

Объясняется все тривиально - при использовании объекта функции компилятор способен **встроить** передаваемую функцию **в тело алгоритма (inline)**, а при использовании обычной функции встраивания не производится.

```
bool f (const int& x,const int& y){ return x<y;}          // функция-предикат

// создаем синоним типа - указатель на функцию сравнения
typedef     bool (*pf) (const int&  ,const int&  ) ;

struct pred{                                    // функтор
      bool operator () (const int& x, const int& y){ return x>y; }
};
```

Функтор

Есть большое множество вариантов их практического применения:

- Необходим вызов обычной функции или функции-члена класса.
- Алгоритм может требовать бинарную или унарную функцию.
- Требуется передача в функцию дополнительных параметров.
- Объект, которому принадлежит функция, может быть константным или неконстантным.
- Аргументы в функцию могут передаваться по значению, указателем или по ссылке.
- Функция может быть реализована в классе.
- Реализация может потребовать использования нескольких функций.

any_of

template <class InputIterator, class **UnaryPredicate**>
bool any_of (InputIterator first, InputIterator last, **UnaryPredicate** pred);

Проверяет, соответствует ли какой-либо элемент в диапазоне условию

Возвращает true, если **предикат** pred возвращает true **для любого** из элементов в диапазоне [first,last), и false в противном случае.

Если [first, last) это пустой диапазон, функция возвращает false.

Поведение этого шаблона функции эквивалентно:

```
template<class InputIterator, class UnaryPredicate>
bool any_of (InputIterator first, InputIterator last, UnaryPredicate pred) {
  while ( first!=last ) {
    if (pred (*first) ) return true;
     ++first;
  }
  return false;
}
```

```
#include <iostream>     // std::cout
#include <algorithm>    // std::any_of
#include <array>        // std::array

int main () {
  std::array <int,7>  foo  =  {0,1,-1,3,-3,5,-5};

  if ( std::any_of ( foo.begin(), foo.end(),
                       [ ](int i) {return i<0;}
                     )
     )
    std::cout << "There are negative elements in the range.\n";

  return 0;
}
```

Output:
There are negative elements in the range.

find

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

Можно использовать алгоритм find(), не зная, как именно он реализован, однако определение алгоритма find () иллюстрирует много полезных проектных идей, поэтому оно достойно изучения (Страуструп). Прежде всего, алгоритм find () применяется к последовательности, определенной парой итераторов.

Мы ищем значение val в полуоткрытой последовательности [first: last). Результат, возвращаемый функцией find (), является итератором. Он указывает либо на первый элемент последовательности, равный значению val, либо на элемент last. Возвращение итератора на элемент, следующий за последним элементом последовательности, самый распространенный способ, с помощью которого алгоритмы библиотеки STL сообщают о том, что элемент не найден.

```
#include <iostream>
#include <algorithm>
#include <vector>
int main () {
  // using std::find with array and pointer:
  int myints[] = { 10, 20, 30, 40 };
  int * p;
       p = std::find (myints, myints+4, 30);
  if (p != myints+4)
    std::cout << "Element found in myints: "
      << *p << '\n';
  else
    std::cout <<
 "Element not found in myints\n";
  // using std::find with vector and iterator:
  std::vector<int> myvector
     (myints,myints+4);
  std::vector<int>::iterator it;
   it = find (myvector.begin(), myvector.end(),
30);
   if (it != myvector.end())
     std::cout << "Element found in myvector:
" << *it << '\n';
   else
     std::cout << "Element not found in
myvector\n";
   return 0;}
```

Output:
Element found in myints: 30
Element found in myvector: 30

adjacent_find

```
template< class ForwardIt >
ForwardIt adjacent_find( ForwardIt first, ForwardIt last );
(1)
template< class ForwardIt, BinaryPredicate p >
ForwardIt adjacent_find( ForwardIt first, ForwardIt last, BinaryPredicate p );
(2)
```

Ищет в диапазоне [first, last) два одинаковых смежных элемента.

- Первый вариант использует operator== для сравнения элементов,
- Второй вариант использует заданный **бинарный предикат p.**

```
#include <iostream>
#include <algorithm>
#include <vector>
bool myfunction (int i, int j) {
  return (i==j);
}
int main () {
  int myints[ ] = {5,20,5,30,30,20,10,10,20};
  std::vector<int> myvector (myints,myints+8);
  std::vector<int>::iterator it;
it = std::adjacent_find (myvector.begin(), myvector.end());
  if (it!=myvector.end())
    std::cout << "the first pair of repeated elements are: " << *it << '\n';
it = std::adjacent_find (++it, myvector.end(), myfunction);
  if (it!=myvector.end())
    std::cout << "the second pair of repeated elements are: " << *it << '\n';
  return 0;
}
```

Output:
the first pair of repeated elements are: 30
the second pair of repeated elements are: 10

std::mismatch

Возвращает первую пару несовпадающих элементов из двух диапазонов: одного, определяемого [first1, last1), и другого, начинающегося с first2.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <utility>          // std::pair

bool mypredicate (int i, int j) {
  return (i==j);
}

int main () {
  std::vector<int> myvector;
   for (int i=1; i<6; i++) myvector.push_back (i*10);  // myvector: 10 20 30 40 50
   int myints[ ] = {10,20,80,320,1024};                //   myints: 10 20 80 320 1024
  std::pair<std::vector<int>::iterator, int*> mypair;
```

```
  // using default comparison:
  mypair = std::mismatch (myvector.begin(), myvector.end(), myints);
  std::cout << "First mismatching elements: " << *mypair.first;
  std::cout << " and " << *mypair.second << '\n';

  ++mypair.first; ++mypair.second;

  // using predicate comparison:
   mypair = std::mismatch ( mypair.first,  myvector.end(),  mypair.second   ,
     mypredicate);
  std::cout << "Second mismatching elements: " << *mypair.first;
  std::cout << " and " << *mypair.second << '\n';

  return 0;
}
```

Output:
First mismatching elements: 30 and 80
Second mismatching elements: 40 and 320

std::search

2. template< class ForwardIt1, class ForwardIt2, class BinaryPredicate >
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last,
ForwardIt2 s_first, ForwardIt2 s_last, BinaryPredicate p);

Ищет первое вхождение последовательности элементов [s_first, s_last) в диапазон [first, last - (s_last - s_first)).

Первый вариант использует operator== для сравнения элементов, второй вариант использует заданный бинарный предикат p.

```
#include <iostream>
#include <algorithm>    // std::search
#include <vector>
bool mypredicate (int i, int j) {
  return (i==j);
}
int main () {
  std::vector<int> haystack;
  // set some values:        haystack: 10 20 30 40 50 60 70 80 90
 for (int i=1; i<10; i++) haystack.push_back(i*10);
```

```
      // using default comparison:
 int needle1[ ] = {40,50,60,70};
 std::vector<int>::iterator it;
 it = std::search (haystack.begin(), haystack.end(), needle1, needle1+4);
if (it!=haystack.end())
   std::cout << "needle1 found at position " << (it-haystack.begin()) << '\n';
 else
   std::cout << "needle1 not found\n";

 // using predicate comparison:
     int needle2[ ] = {20,30,50};
 it = std::search (haystack.begin(), haystack.end(), needle2, needle2+3, mypredicate);

 if ( it!=haystack.end() )
   std::cout << "needle2 found at position " << (it-haystack.begin()) << '\n';
 else
   std::cout << "needle2 not found\n";

 return 0;  }
```

Output:
needle1 found at position 3
needle2 not found

std::copy

template< class InputIt, class OutputIt >
OutputIt copy(InputIt first, InputIt last, OutputIt d_first); (1)
Копирует элементы диапазона [first, last) в диапазон, начинающийся с d_first.

template< class InputIt, class OutputIt, class UnaryPredicate >
OutputIt copy_if(InputIt first, InputIt last,
OutputIt d_first,
UnaryPredicate pred); (2) (начиная с C++11)
Копирует только те элементы, для которых предикат pred возвращает true.
Определение функции предиката должно быть эквивалентно следующему:
bool pred(const Type &a);

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <iterator>
     int main() {
    std::vector<int> from_vector;
    for (int i = 0; i < 10; i++) {
        from_vector.push_back(i);
    }
     std::vector<int> to_vector(10);
     std::copy(from_vector.begin(), from_vector.end(), to_vector.begin());
     std::cout << "to_vector содержит: \n";
    std::copy(to_vector.begin(), to_vector.end(),
              std::ostream_iterator<int>(std::cout, " "));
    std::cout << std::endl;
}
```

Output:
to_vector содержит:
0 1 2 3 4 5 6 7 8 9

std::transform

```
template< class InputIt, class OutputIt, class UnaryOperation >
OutputIt transform( InputIt first1, InputIt last1, OutputIt d_first,
                    UnaryOperation unary_op );                          (1)

template< class InputIt1, class InputIt2, class OutputIt, class BinaryOperation >
OutputIt transform( InputIt1 first1, InputIt1 last1, InputIt2 first2,
                    OutputIt d_first, BinaryOperation binary_op );      (2)
```

Применяет заданную функцию к одному диапазону и сохраняет результат в другой диапазон, начинающийся с d_first.

В первом варианте унарная операция unary_op применяется к диапазону [first1, last1). Во втором варианте бинарная операция binary_op применяется к элементам из двух диапазонов: [first1, last1) и начинающемуся с first2

```
Ret fun(const Type &a);
Ret fun(const Type1 &a, const Type2 &b);
```

```
#include <iostream>     // std::cout
#include <algorithm>    // std::transform
#include <vector>       // std::vector
#include <functional>   // std::plus
     int op_increase (int i) { return ++i; }
int main () {
  std::vector<int> foo;
  std::vector<int> bar;
for (int i=1; i<6; i++)
   foo.push_back (i*10);              // foo: 10 20 30 40 50
   bar.resize(foo.size());            // allocate space
  std::transform (foo.begin(), foo.end(), bar.begin(), op_increase);
                                      // bar: 11 21 31 41 51
  // std::plus adds together its two arguments:
  std::transform (foo.begin(), foo.end(), bar.begin(), foo.begin(), std::plus<int>());
                                      // foo: 21 41 61 81 101
  std::cout << "foo contains:";
  for (std::vector<int>::iterator it=foo.begin(); it!=foo.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
  return 0;}
```

Output:
foo contains: 21 41 61 81 101

std::remove

```
template< class ForwardIt, class T >
ForwardIt remove( ForwardIt first, ForwardIt last, const T& value );        (1)

template< class ForwardIt, class UnaryPredicate >
ForwardIt remove_if( ForwardIt first, ForwardIt last, UnaryPredicate p );  (2)
```

Удаляет из диапазона [first, last) все элементы, удовлетворяющие определенному условию. Первый вариант удаляет все элементы, равные value, второй вариант удаляет все элементы, для которых **предикат** p возвращает true.

Удаление осуществляется путём сдвига элементов внутри диапазона таким образом, что удаляемые элементы перезаписываются. Элементы между старым и новым концами диапазона имеют неопределённое значение. Возвращается итератор на новый конец диапазона. Относительный порядок оставшихся элементов сохраняется

Определение функции **предиката** должно быть эквивалентно следующему:

```
bool pred ( const Type &a );
```

```
#include <algorithm>
#include <string>
#include <iostream>

int main()
{
    std::string str = "Текст с несколькими   пробелами";
    str.erase(std::remove(str.begin(), str.end(), ' '),   str.end());
    std::cout << str << '\n';
}
```

Output:

Текстснесколькимипробелами

std::rotate

```
template< class ForwardIt >
void rotate( ForwardIt first, ForwardIt n_first, ForwardIt last );
template< class ForwardIt >
ForwardIt rotate( ForwardIt first, ForwardIt n_first, ForwardIt last );
```

Меняет местами (вращает) элементы в диапазоне [first, last) таким образом, что элемент n_first становится первым в новом диапазоне, а n_first-1 — последним.

Пример

```
#include <vector>
#include <iostream>
#include <algorithm>
 std::ostream& operator<<(std::ostream& s, const std::vector<int>& v){
    for(int n: v)        s << n << ' ';
    return s;
}
 int main() {
    std::vector<int> v{2, 4, 2, 0, 5, 10, 7, 3, 7, 1};
    std::cout << "до сортировки   :      " << v << std::endl;
          // сортировка вставками
    for (auto i = v.begin(); i != v.end(); ++i) {
        std::rotate (std::upper_bound (v.begin(), i, *i), i, i+1);
    }
    std::cout << "после сортировки:      " << v << std::endl;
     // вращение влево
    std::rotate (v.begin(), v.begin() + 1, v.end());
    std::cout << "после вращения влево : " << v << std::endl;
     // вращение вправо
    std::rotate (v.rbegin(), v.rbegin() + 1, v.rend());
    std::cout << "после вращения вправо: " << v << std::endl;
}
```

Output:
до сортировки : 2 4 2 0 5 10 7 3 7 1
после сортировки: 0 1 2 2 3 4 5 7 7 10
после вращения влево : 1 2 2 3 4 5 7 7 10 0
после вращения вправо: 0 1 2 2 3 4 5 7 7 10

std::sort

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );                    (1)
template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );  (2)
```

Сортировка элементов в диапазоне [first, last) в порядке возрастания. Сохранность порядка элементов, имеющих одинаковое значение, не гарантируется.

Для сравнения элементов по умолчанию используется оператор operator<, но также может быть использована функция сравнения cmp.

Сигнатура функции(предиката) сравнения должна быть эквивалентна следующей:

```
bool  cmp (const Type1 &a,   const Type2 &b);
```

```
#include <algorithm>
#include <functional>
#include <array>
#include <iostream>
#include <iterator>
 int main(){
    std::array<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

    std::sort(s.begin(), s.end());
    std::copy(s.begin(), s.end(), std::ostream_iterator<int>(std::cout," "));
    std::cout << std::endl;

    std::sort(s.begin(), s.end(), std::greater<int>());
    std::copy(s.begin(), s.end(), std::ostream_iterator<int>(std::cout," "));
    std::cout << std::endl;
}
```

Output:

0 1 2 3 4 5 6 7 8 9

9 8 7 6 5 4 3 2 1 0

lower_bound

template< class ForwardIt, class T >

ForwardIt **lower_bound**(ForwardIt first, ForwardIt last, const T& value); (1)

template< class ForwardIt, class T, class Compare >

ForwardIt **lower_bound**(ForwardIt first, ForwardIt last, const T& value, Compare comp); (2)

Возвращает итератор на первый элемент диапазона [first, last) не меньший (**равный или больший**) чем value.

Первая версия использует operator< для сравнения элементов, вторая версия использует переданную функцию сравнения comp.

upper_bound

template< class ForwardIt, class T >
ForwardIt **upper_bound**(ForwardIt first, ForwardIt last, const T& value); (1)
template< class ForwardIt, class T, class Compare >
ForwardIt **upper_bound**(ForwardIt first, ForwardIt last, const T& value, Compare comp); (2)

Возвращает итератор, указывающий на первый элемент в диапазоне [first, last) то есть **больше**, чем value.

Первый вариант используется operator< для сравнения элементов, вторая версия использует данную функцию сравнения comp.

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

int main()
{
    std::vector<int> data = { 1, 1, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 6 };

    auto lower = std::lower_bound(data.begin(), data.end(), 4);
    auto upper = std::upper_bound(data.begin(), data.end(), 4);

    std::copy(lower, upper, std::ostream_iterator<int>(std::cout, " "));
}
```

Output:
4 4 4

equal_range

```
template< class ForwardIt,  class T >
std::pair < ForwardIt, ForwardIt >
    equal_range( ForwardIt first, ForwardIt last,
                 const T& value );                                  (1)
template< class ForwardIt, class T, class Compare >
std::pair <ForwardIt, ForwardIt >
    equal_range( ForwardIt first, ForwardIt last,
                 const T& value, Compare comp );                    (2)
```

Возвращает диапазон, содержащий все элементы равные value в отсортированном диапазоне [first, last).

Диапазон определяется двумя итераторами, первый указывает на первый элемент не меньший (**равный или больше**), чем value, второй указывает на первый элемент **больший**, чем value.

Первый итератор может быть получен использованием lower_bound(), второй - upper_bound().

Первый вариант для сравнения элементов использует operator<, второй вариант использует передаваемую функцию сравнения comp.

```
#include <algorithm>
#include <vector>
#include <iostream>
     struct S {
    int number;
    char name;
         S ( int number, char name  ) : number ( number ), name ( name )   { }
    bool operator< ( const S& s ) const    {
       return number < s.number;
    }
};

 int main(){
      std::vector<S> vec = { {1,'A'}, {2,'B'}, {2,'C'}, {2,'D'}, {3,'F'}, {4,'G'} };
     S value ( 2, '?' );
     auto p = std::equal_range(vec.begin(),vec.end(),value);
     for ( auto i = p.first; i != p.second; ++i )
       std::cout << i->name << ' ';
}
```

Output:

B C D

set_intersection

```
template< class InputIt1, class InputIt2, class OutputIt >
OutputIt set_intersection( InputIt1 first1, InputIt1 last1,
                           InputIt2 first2, InputIt2 last2,
                           OutputIt d_first );                          (1)
template< class InputIt1, class InputIt2,
          class OutputIt, class Compare >
OutputIt set_intersection( InputIt1 first1, InputIt1 last1,
                           InputIt2 first2, InputIt2 last2,
                           OutputIt d_first, Compare comp );            (2)
```

Создает отсортированный диапазон начало в d_first, состоящей из элементов, которые встречаются в обоих диапазонах отсортированы [first1, last1) и [first2, last2).

Первая версия ожидает, что обе входные диапазоны должны быть отсортированы operator<, вторая версия ожидает, что они должны быть отсортированы данной comp функцией сравнения.

Если некоторый элемент не найден m раз в [first1, last1) и n раз в [first2, last2), первые элементы std::min (m, n) будут скопированы из первого диапазона в диапазон назначения.

Порядок эквивалентных элементов сохраняется.

Результирующий диапазон не должен пересекаться с любым из входных диапазонов.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
int main()
{
    std::vector<int> v1{1,2,3,4,5,6,7,8};
    std::vector<int> v2{           5,   7,   9,10};
    std::sort(v1.begin(), v1.end());
    std::sort(v2.begin(), v2.end());

    std::vector<int> v_intersection;

    std::set_intersection(v1.begin(), v1.end(),
                          v2.begin(), v2.end(),
                          std::back_inserter ( v_intersection));
    for( int n : v_intersection)
        std::cout << n << ' ';
}
```

Output:

5 7

set_union

```
template< class InputIt1, class InputIt2, class OutputIt >
OutputIt set_union( InputIt1 first1, InputIt1 last1,
                    InputIt2 first2, InputIt2 last2,
                    OutputIt d_first );                    (1)
template< class InputIt1, class InputIt2,
          class OutputIt, class Compare >
OutputIt set_union( InputIt1 first1, InputIt1 last1,
                    InputIt2 first2, InputIt2 last2,
                    OutputIt d_first, Compare comp );      (2)
```

Создает отсортированный диапазон начало в d_first, состоящий из всех элементов, присутствующих в одном или обоих диапазонах отсортированы [first1, last1) и [first2, last2).

Первая версия ожидает, что обе входные диапазоны должны быть отсортированы с помощью operator<, вторая версия ожидает, что они должны быть отсортированы данной comp функцией сравнения.

Если некоторый элемент не найден m раз в [first1, last1) и n раз в [first2, last2), то все m элементы будут скопированы из [first1, last1) в d_first, сохраняя порядок, и тогда точно std::max(n-m, 0) элементов будут скопированы из [first2, last2) в d_first, а также сохранен их порядок.

Результирующий диапазон не должен пересекаться с любым из входных диапазонов.

```
#include <iostream>     // std::cout
#include <algorithm>    // std::set_union, std::sort
#include <vector>       // std::vector
int main () {
  int first[ ] = {5,10,15,20,25};
  int second[ ] = {50,40,30,20,10};
  std::vector<int> v(10);                      // 0  0  0  0  0  0  0  0  0  0
  std::vector<int>::iterator it;
  std::sort (first,first+5);     //  5 10 15 20 25
  std::sort (second,second+5);   // 10 20 30 40 50
it=std::set_union (first, first+5, second, second+5, v.begin());
        // 5 10 15 20 25 30 40 50  0  0

  v.resize(it-v.begin());                      // 5 10 15 20 25 30 40 50
  std::cout << "The union has " << (v.size()) << " elements:\n";
  for (it=v.begin(); it!=v.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';
  return 0;
}
```

Output:
The union has 8 elements:
5 10 15 20 25 30 40 50

Темы для проведения исследований.

Общая тематика: исследование поведения в условиях неопределенности.

Темы:

1. Оптимальное поведение в не штатной ситуации (при наводнении). (Организационное поведение, частное поведение, психо-оптимальное поведение)
2. Проектирование программ слабоопределенных целей. (Спутники и наземные станции)
3. Перемена целей в условиях нарушенной (разрушенной) логистики. Проектирование поведения.
4. Моделирование настойчивости (временная характеристика(3), Locke2002) в достижении цели (выбор стратегии, отказ от стратегии в пользу иной, отдых, пережидание)
5. Моделирование ситуаций побуждения (косвенных целей (4), Locke2002). То, что я делаю вызовет окружающие изменения, которые нам нужны.
6. Моделирование познания и мотивации (где?)
7. Визуализация мероприятий планирования (действий, строительства, организации)

Темы для проведения исследований.

8. Визуализация стратегии выбора в бизнесе.

Предмет (что):

- варианты развития
- варианты поведения
- варианты обстоятельств
- варианты откликов на обстоятельства

Метод (как):

- генерация графов решений и поступков

Новизна:

- когнитивное согласование графов решений и поступков
- выбор и игра с вариантами поведения на протяжении всего ЖЦ бизнеса

Польза (актуальность):

- выигрыш от раннего графического представления предполагаемых ситуаций и их разрешения
- выстраивание бизнеса по лекалам визуализированных предположений.

9. Выработка поведенческой стратегии при нарушении логистической цепи переправки грузов.

10. Передвижение животных.

11. Когнитивная визуализация помощи (подсказки, поддержки, совета)

12. Движение бактерий (моделирование путей их распространения)

13. Прогнозирование социального поведения на основе малозаметных деталей, реакций (исследование агрессий)

14. Визуализация процесса разработки ПО (сочетание планов, блок-схем, стратегий решений) + автоматизация поиска с отображением результатов, сортировок, тенденций + когнитивные советы. (цель: актуализация понимания как всего процесса разраотки ПО, на макроуровне — время место, будущее, так и частных потребностей и решений — что есть и что нужно).

15. Риск: рисковые стратегии, снижение риска, оценки рисков, риск поведения, риск движения.

16. Беспилотный автомобиль. Оценка рисков в многообразии ситуаций. С учетом психологического фактора.

ДЗ

Продолжаем прошлый проект.