Языки программирования

Standard Template Library, часть 3

Итераторы

Для любого итератора реализованы операции:

- префиксного и постфиксного инкремента (++)
- присвоения (=)
- сравнения на равенство (==) и неравенство (!=)

Категории итераторов

Категория	Операции	Контейнеры
входной (In)	* для чтения значения x = *i;	все
выходной (Out)	*для присвоения значения *i = x;	все
прямой (For)	* для чтения и присвоения значения x = *i; *i = x;	все
двунаправленный (Bi)	как прямой и операция -- --i; i--;	все
произвольного доступа (Ran)	как двунаправленный и операции сложения и вычитания константы i + n, i - n, i += n, i -= n; сравнения i < j, i > j, i <= j, i >= j	все, кроме list

Действительные и недействительные итераторы

Действительный итератор указывает на какой-либо элемент контейнера.

Итератор **недействителен**, если:

- он не был инициализирован
- контейнер, с которым он связан, изменил размеры или удален
- итератор указывает на конец последовательности

Другие разновидности итераторов

- **Обратные итераторы** – перечисляют элементы контейнера в обратном порядке

```
for (vector<int>::reverse_iterator i = vec.rbegin();
 i != vec.rend(); i++)
   cout << *i << " ";
```

Другие разновидности итераторов (продолжение)

- **Итераторы вставки** – позволяют вставлять в контейнер **x** типа **C** значения из алгоритмов. Определены три шаблонные функции, создающие итераторы вставки:
  - в конец

```
template <class C>
back_insert_iterator<C> back_inserter(C& x);
```

  - в начало

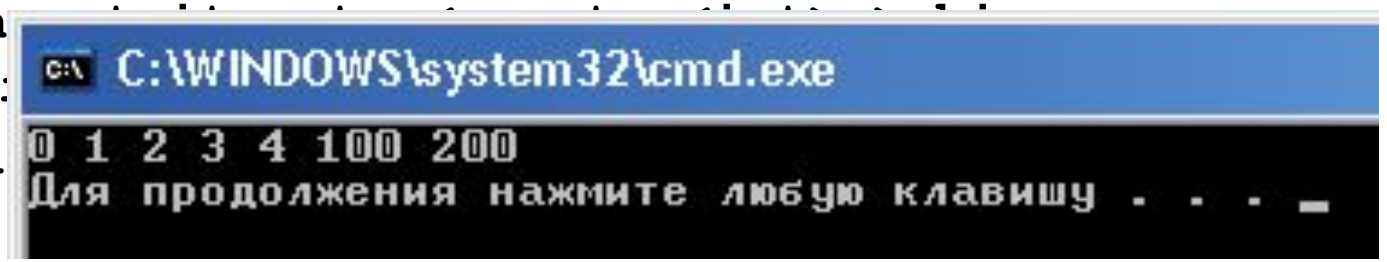
```
template <class C>
front_insert_iterator<C> front_inserter(C& x);
```

  - в позицию итератора **i** контейнера

```
template <class C, class Iter>
insert_iterator<C> inserter(C& x, Iter i);
```

Пример использования итераторов вставки

```
vector<int> vec;
for (int i = 0; i < 5; i++)
  vec.push_back(i);
back_in
  back_
*bi = 1
bi++;
*bi = 200; //вставка
for (int i = 0; i < vec.size(); i++)
  cout << vec[i] << " ";
cout << endl;
```



Другие разновидности итераторов (продолжение)

- **Потоковые итераторы** – позволяют стандартным алгоритмам использовать потоки ввода/вывода. Определены два шаблонных класса:
  - итератор ввода

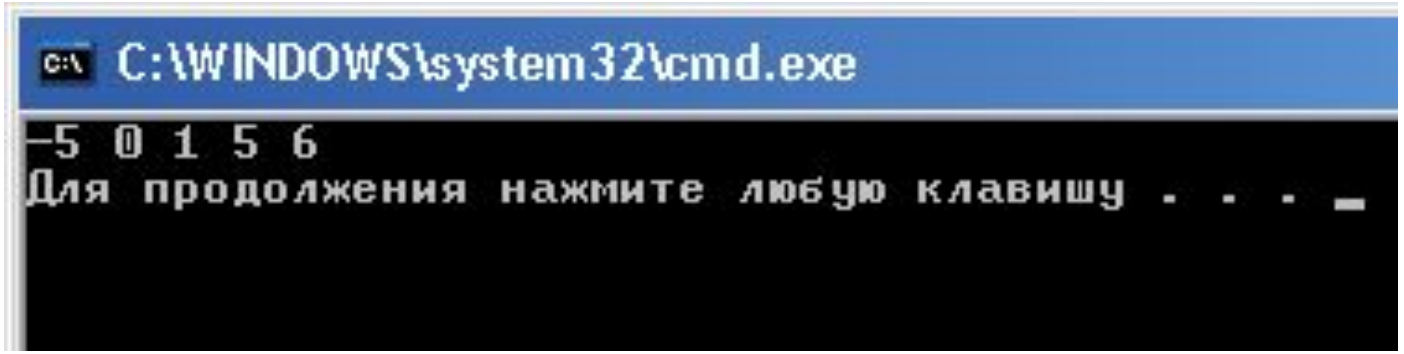
```
ifstream inp("input.txt");
istream_iterator<int> i(inp);
int a = *i; //чтение первого числа из файла
i++;
int b = *i; //чтение второго числа из файла
```

  - итератор вывода

```
ostream out("output.txt");
ostream_iterator<int> os(out, " ");
*os = 555; //выводит 555 и пробел
os++;
*os = 666; //выводит 666 и пробел
```

Вместо итераторов можно работать с обычными указателями

```
const int N = 5;
int mas [N] = { 6, 1, 5, 0, -5};
sort(mas, mas + N); //Алгоритм сортировки
for (int i = 0; i < N; i++)
  cout << mas[i] << " ";
cout << endl;
```



Функциональные объемы

Функциональные объекты – классы, в которых определена операция вызова ().

- Везде, где в стандартных алгоритмах можно использовать функциональный объект, можно использовать и указатель на функцию.
- Для работы со стандартными функциональными объектами необходимо подключить заголовочный файл **<functional>**

Виды функциональных объектов

- **Арифметические** – позволяют выполнять стандартные арифметические операции
- **Предикаты** – функциональные объекты, возвращающие значение типа bool

Дополнительные средства STL:

- **Отрицатели** – используются для инверсии значений предиката
- **Связыватели** – предназначены для использования функционального объекта с двумя аргументами как объекта с одним аргументом
- **Адаптеры указателей на функцию** – позволяют использовать указатели на функцию в качестве функциональных объектов
- **Адаптеры методов** – позволяют использовать методы классов в качестве функциональных объектов

Арифметические функциональные объекты

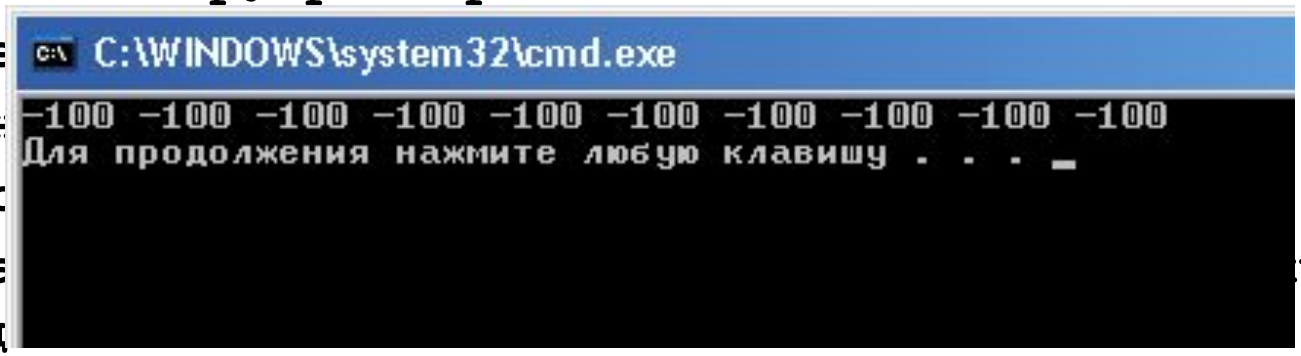
Имя	Тип	Результат
plus	бинарный	x + y
minus	бинарный	x - y
multiplies	бинарный	x * y
divides	бинарный	x / y
modulus	бинарный	x % y
negate	унарный	- x

Описание реализации plus внутри STL

```
template <class T>
struct plus : binary_function<T, T, T>
{
  T operator() (const T& x, const T& y) const
   {
   return x + y
   }
};
```

Пример использования унарного функционального объекта negate

```
vector<int> a;
//Задаем вектору размер 10
a.resize
//Заполн
fill(a.b
//Над ка                                          ую
  операц
//результат помещаем в вектора a
transform(a.begin(), a.end(), a.begin(), negate<int>());
for (int i = 0; i < a.size(); i++)
  cout << a[i] << " ";
cout << endl;
```



Предикаты

Имя	Тип	Результат
equal_to	бинарный	x == y
not_equal_to	бинарный	x != y
greater	бинарный	x > y
less	бинарный	x < y
greater_equal	бинарный	x >= y
less_equal	бинарный	x <= y
logical_and	бинарный	x && y
logical_or	бинарный	x || y
logical_not	унарный	! x

Описание реализации equal_to внутри STL

```
template <class T>
struct equal_to : binary_function<T, T, bool>
{
  bool operator() (const T& x, const T& y) const
  {
   return x == y;
  }
};
```

Отрицатели

Шаблонные функции для получения противоположного значения:

- унарного предиката

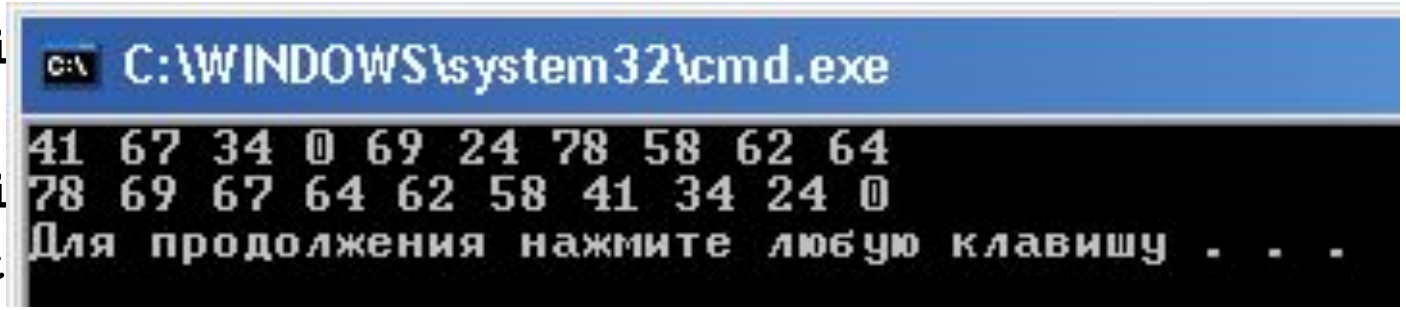
```
template <class Predicate>
unary_negate<Predicate> not1(const Predicate& pred);
```

- бинарного предиката

```
template <class Predicate>
binary_negate<Predicate> not2(const Predicate& pred);
```

Пример использования отрицателя

```
vector<int> a;
a.resize(10);
for (i
  a[i]
for (i
  cout
cout << endl;
//Сортируем в порядке >= (не убывания)
sort(a.begin(), a.end(), not2(less<int>()));
for (int i = 0; i < a.size(); i++)
  cout << a[i] << " ";
cout << endl;
```



Алгоритмы STL

- Предназначены для работы с разнообразными контейнерами STL и другими последовательностями
- Каждый алгоритм реализован в виде одной или нескольких шаблонных функций
- Алгоритмам передаются итераторы, определяющие начало и конец обрабатываемой последовательности. Вид итераторов определяет типы контейнеров, для которых может использоваться алгоритм. Например, алгоритму **`sort`** необходимы итераторы произвольного доступа, поэтому он не будет работать с **`list`**
- Алгоритмы не выполняют проверку выхода за пределы последовательности
- Для настройки алгоритмов используются функциональные объекты
- Необходимо подключать заголовочный файл **`<algorithm>`**

Виды алгоритмов

- не модифицирующие операции с последовательностями
- модифицирующие операции с последовательностями
- алгоритмы, связанные с сортировкой
- алгоритмы работы с множествами и пирамидами (кучами)

Общие принципы наименования алгоритмов

- (алгоритм)_if – алгоритм использует предикат для анализа элементов контейнера
- (алгоритм)_n – алгоритм использует заданное количество элементов/значений
- (алгоритм)_copy – алгоритм перед работой копирует часть исходного контейнера в другой контейнер

Используемые обозначения

- In – итератор для чтения
- Out – итератор для записи
- For – прямой итератор
- Bi – двунаправленный итератор
- Ran – итератор произвольного доступа
- Pred – унарный предикат
- BinPred – бинарный предикат
- Op – унарная операция
- BinOp – бинарная операция

Не модифицирующие операции с последовательностями

Алгоритм	Выполняемая функция
adjacent_find	Нахождение пары соседних значений
count	Подсчет количества вхождений значения в последовательность
count_if	Подсчет количества выполнений условия в последовательности
equal	Попарное равенство элементов двух последовательностей
find	Нахождение первого вхождения значения в последовательность
find_end	Нахождение последнего вхождения одной последовательности в другую
find_first_of	Нахождение первого значения из одной последовательности в другой
find_if	Нахождение первого соответствия условию в последовательности
for_each	Вызов функции для каждого элемента последовательности
mismatch	Нахождение первого несовпадающего элемента в двух последовательностях
search	Нахождение первого вхождения одной последовательности в другую
search_n	Нахождение n-го вхождения одной последовательности в другую

adjacent_find – нахождение пары равных соседних значений последовательности

- С использованием == для сравнения элементов. Возвращаемое значение - итератор на первое из них. Если все различны – итератор на элемент за концом последовательности:

```
template <class For>
For adjacent_find(For first, For last);
```

- С использованием бинарного предиката для сравнения элементов:

```
template <class For, class BinPred)
For adjacent_find(For first, For last,
          BinPred pred);
```

count, count_if – подсчет количества вхождений в последовательность

- Заданного значения **value (==)**

```
template <class In, class T>
difference_type count(In first, In last,
                      const T& value);
```

- Удовлетворяющих унарному предикату

```
template <class In, class Pred>
difference_type count_if(In first, In last,
                         Pred pred);
```

equal – поэлементное равенство двух последовательностей

- Поэлементная проверка с использованием ==

```
template <class In1, class In2>
bool equal(In1 first1, In1 last1, In2 first2);
```

- Проверка с использованием бинарного предиката для каждой пары элементов

```
template <class In1, class In2, class BinPred>
bool equal(In1 first1, In1 last1, In2 first2,
     BinPred pred);
```

find, find_if – нахождение первого вхождения в последовательность

- Значения **value (==)**. Возвращаемое значение – итератор на вхождение.

```
template <class In, class T>
In find(In first, In last, const T& value);
```

- Удовлетворяющего унарному предикату

```
template <class In, class Pred>
In find_if(In first, In last, Pred pred);
```

find_first_of – нахождение первого вхождения элементов из одной последовательности в другой

- С использованием **==** для проверки равенства. **[first1, last1)** – где ищем, **[first2, last2)** – что ищем. Возвращаемое значение – итератор на вхождение. Не найдено => возвращает **last1**.

```
template <class For1, class For2>
For1 find_first_of(For1 first1, For1 last1,
                   For2 first2, For2 last2);
```

- С использованием бинарного предиката

```
template <class For1, class For2, class BinPred>
For1 find_first_of(For1 first1, For1 last1,
                   For2 first2, For2 last2,
                   BinPred pred);
```

find_end - нахождение последнего вхождения элементов из одной последовательности в другой

- С использованием **==** для проверки равенства. **[first1, last1)** – где ищем, **[first2, last2)** – что ищем. Возвращаемое значение – итератор на вхождение. Не найдено => возвращает **last1**.

```
template <class For1, class For2>
For1 find_end(For1 first1, For1 last1,
        For2 first2, For2 last2);
```

- С использованием бинарного предиката

```
template <class For1, class For2, class BinPred>
For1 find_end(For1 first1, For1 last1,
        For2 first2, For2 last2,
        BinPred pred);
```

for_each – вызов функции для каждого элемента последовательности

- Параметр f – унарная операция

```
template <class In, class Function>
For1 for_each(In first, In last, Function f);
```

mismatch – нахождение первого несовпадающего элемента в двух последовательностях

- Поэлементная проверка с помощью ==. Возвращается пара-структура, содержащее два поля: first – итератор на отличающийся элемент в первой последовательности, second – во второй.

```
template <class In1, class In2>
pair<In1, In2> mismatch(In1 first1, In1 last1,
                        In2 first2);
```

- Проверка с использованием бинарного предиката для каждой пары элементов

```
template <class In1, class In2, class BinPred>
 pair<In1, In2> mismatch(In1 first1, In1 last1,
                         In2 first2, BinPred pred);
```

search- нахождение первого вхождения в последовательность другой последовательности в качестве подпоследовательности

- С использованием == для сравнения элементов. [first1, last1) – где икать, [first2, last2) – что искать. Возвращаемое значение – итератор на вхождение.

```
template <class For1, class For2>
For1 search(For1 first1, For1 last1,
      For1 first2, For2 last2);
```

- С использованием бинарного предиката для сравнение элементов

```
template <class For1, class For2, class BinPred>
For1 search(For1 first1, For1 last1,
      For1 first2, For2 last2,
      BinPred pred);
```

search_n – нахождение первого вхождения в последовательность подпоследовательности, состоящей из n значений value

- С использованием == для сравнения элементов. Возвращаемое значение – итератор на вхождение.

```
template <class For, class Size, class T>
For1 search_n(For first, For1 last,
        Size n, const T& value);
```

- С использованием бинарного предиката для сравнения элементов

```
template <class For, class Size, class T,
              class BinPred>
For1 search_n(For first, For1 last,
        Size n, const T& value,
        BinPred pred);
```

Модифицирующие операции с последовательностями

Алгоритм	Выполняемая функция
copy	Копирование последовательности, начиная с первого элемента
copy_backward	Копирование последовательности, начиная с последнего элемента
fill	Замена всех элементов заданным значением
fill_n	Замена первых n элементов заданным значением
generate	Замена всех элементов результатом операции
generate_n	Замена первых n элементов результатом операции
iter_swap	Обмен местами двух элементов, заданных итераторами
random_shuffle	Перемещение элементов в соответствии со случайным равномерным распределением
remove	Перемещение элементов с заданным значением
remove_copy	Копирование последовательности с перемещением элементов с заданным значением
remove_copy_if	Копирование последовательности с перемещением элементов при выполнении предиката
remove_if	Перемещение элементов при выполнении предиката

Модифицирующие операции с последовательностями (продолжение)

replace_copy_if	Копирование последовательности с заменой элементов при выполнении предиката
replace_if	Замена элементов при выполнении предиката
reverse	Изменение порядка элементов на обратный
reverse_copy	Копирование последовательности в обратном порядке
rotate	Циклическое перемещение элементов последовательности
rotate_copy	Циклическое копирование элементов
swap	Обмен местами двух элементов
swap_ranges	Обмен местами элементов двух последовательностей
transform	Выполнение заданной операции над каждым элементом последовательности
unique	Удаление равных соседних элементов
unique_copy	Копирование последовательности с удалением равных соседних элементов

copy, copy_backward – копирование элементов последовательности в выходную последовательность

- Копирование, начиная с первого элемента последовательности. **[first, last)** – откуда копируем, **result**-куда

```
template <class In, class Out>
Out copy(In first, In last, Out result);
```

- Копирование, начиная с последнего элемента последовательности

```
template <class Bi1, class Bi2>
Bi2 copy_backward(Bi1 first, Bi1 last,
          Bi2 result);
```

fill, fill_n – замена элементов последовательности заданным значением

- на интервале **[first, last)**.**value** - значение

```
template <class For, class T>
void fill(For first, For last, const T& value);
```

- только первых **n** значений, начиная с **first**

```
template <class Out, class Size, class T>
void fill_n(Out first, Size n, const T& value);
```

generate, generate_n – замена элементов последовательности значениями функции-генератора

- на интервале **[first, last).gen** – функциональный объект (содержит операцию () без параметров), возвращающий очередное значение

```
template <class For, class Generator>
void generate(For first, For last, Generator gen);
```

- только первых **n** значений, начиная с **first**

```
template <class Out, class Size, class Generator>
void fill_n(Out first, Size n, Generator gen);
```

iter_swap, swap, swap_ranges – обмен местами

- двух элементов, заданных итераторами **a** и **b**

```
template <class For1, class For2>
void iter_swap(For1 a, For2 b);
```

- двух элементов, заданных ссылками **a** и **b**

```
template <class T>
void swap(T& a, T& b);
```

- элементов двух диапазонов с одинаковым числом элементов.

[first1, last1) – первый диапазон, **first2** – начало второго

```
template <class For1, class For2>
For2 swap_ranges(For1 first1, For1 last1,
                 For2 first2);
```

random_shuffle – случайная перестановка элементов последовательности

- на интервале **[first, last)** с использованием равномерного распределения

```
template <class Ran>
void random_shuffle(Ran first, Ran last);
```

- на интервале **[first, last)** с использованием собственного генератора случайных чисел **rgen**, описанного в виде функционального объекта (с операцией () без параметров для получения очередного случайного значения)

```
template <class Ran, class RandomGenerator>
void random_shuffle(Ran first, Ran last,
                    RandomGenerator& rgen);
```

remove, remove_if – перемещение в конец интервала последовательности всех элементов

- равных **value**. **[first, last)** – интервал. Все оставшиеся элементы сохраняют относительный порядок, возвращает границу их размещения

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

- удовлетворяющих унарному предикату **pred**

```
template <class For, class Pred>
For remove_if(For first, For last, Pred pred);
```

remove_copy, remove_copy_if – перемещение в конец интервала копии последовательности всех элементов

- Отличаются от предыдущих алгоритмов тем, что перед обработкой последовательность копируется на место, задаваемое итератором result

```
template <class For, class Out, class T>
For remove_copy(For first, For last,
          Out result, const T& value);

template <class For, class Out, class Pred>
For remove_copy_if(For first, For last,
          Out result, Pred pred);
```

replace, replace_if – замена элементов последовательности

- равных **old_value** на значение **new_value**. **[first, last)** – интервал.

```
template <class For, class T>
void replace(For first, For last,
        const T& old_value,
        const T& new_value);
```

- удовлетворяющих унарному предикату **pred** на значение **new_value**

```
template <class For, class Pred>
void replace_if(For first, For last, Pred pred,
            const T& new_value);
```

replace_copy, replace_copy_if – замена элементов в копии последовательности

- Отличаются от предыдущих алгоритмов тем, что перед обработкой последовательность копируется на место, задаваемое итератором **result**

```
template <class For, class Out, class T>
void replace(For first, For last, Out result,
       const T& old_value,
       const T& new_value);

template <class For, class Out, class Pred>
void replace_if(For first, For last, Out result,
          Pred pred, const T& new_value);
```

reverse, reverse_copy – изменение порядка элементов последовательности на обратный

- в исходной последовательности. **[first, last)** – интервал

```
template <class Bi>
void reverse(Bi first, Bi last);
```

- в копии исходной последовательности. **result** – место, куда производится предварительное копирование

```
template <class Bi, Out result>
void reverse(Bi first, Bi last, Out result);
```

rotate, rotate_copy – циклическое перемещение элементов последовательности

- в исходной последовательности. **[first, last)** – интервал, **middle** – элемент, который должен стать первым после перемещения

```
template <class For>
void rotate(For first, For middle, For last);
```

- в копии исходной последовательности. **result** – место, куда производится предварительное копирование

```
template <class For, class Out>
void rotate(For first, For middle, For last,
      Out result);
```

transform – выполнение операции для

- каждого элемента исходной последовательности. **[first, last)** – интервал, **op** – унарная операция, **result** – начало результирующей последовательности. Возвращаемое значение – итератор на следующий элемент после последнего результата в результирующей последовательности

```
template <class In, class Out, class Op>
Out transform(In first, In last, Out result,
        Op op);
```

- каждой пары соответствующих элементов двух последовательностей. **[first1, last1)** – интервал первой последовательности, **first2** – начало второй последовательности, **bin_op** – бинарная операция

```
template <class In1, class In2 class Out,
       class Op>
Out transform(In1 first1, In1 last1, In2 first,
        Out result, BinOp bin_op);
```

unique – удаление повторяющихся соседних элементов

- проверка с помощью операции ==. Возвращаемое значение – итератор на новый логический конец данных. При удалении размер контейнера не меняется!

```
template <class For>
For unique(For first, For last);
```

- проверка элементов на равенство осуществляется с помощью бинарного предиката **bin_pred**

```
template <class For, class BinPred>
For unique(For first, For last,
           BinPred bin_pred);
```

unique_copy – удаление повторяющихся соседних элементов в копии последовательности

- Отличаются от предыдущих алгоритмов тем, что перед обработкой последовательность копируется на место, задаваемое итератором **result**

```
template <class For, class Out>
Out unique_copy(For first, For last,
                Out result);

template <class For, class Out, class BinPred>
Out unique_copy(For first, For last, Out result,
                BinPred bin_pred);
```

Алгоритмы, связанные с сортировкой

Алгоритм	Выполняемая функция
binary_search	Поиск заданного значения
equal_range	Нахождение последовательности элементов с заданным значением
inplace_merge	Слияние отсортированных последовательностей одного диапазона
lexicographical_compare	Лексикографически первая из двух последовательностей
lower_bound	Нахождение первого вхождения заданного значения
max	Большее из двух значений
max_element	Наибольшее значение в последовательности
merge	Слияние отсортированных последовательностей
min	Меньшее из двух значений
min_element	Наименьшее значение в последовательности

Алгоритмы, связанные с сортировкой (продолжение)

next_permutation	Следующая перестановка в лексикографическом порядке
nth_element	Помещение n-го элемента на заданное место
partial_sort	Частичная сортировка
partial_sort_copy	Частичная сортировка с копированием
partition	Перемещение вперед элементов, удовлетворяющих условию
prev_permutation	Предыдущая перестановка в лексикографическом порядке
sort	Сортировка
stable_partition	Перемещение вперед элементов, удовлетворяющих условию, с сохранением их относительного порядка
stable_sort	Сортировка, сохраняющая порядок для одинаковых элементов
upper_bound	Нахождение первого элемента, большего, чем заданное значение

Алгоритмы работы с множествами и пирамидами

Алгоритм	Выполняемая функция
includes	Включение одного множества в другое
set_intersection	Создание отсортированного пересечения множеств
set_difference	Создание отсортированной последовательности элементов, входящих только в первую из двух последовательностей
set_symmetric_difference	Создание отсортированной последовательности элементов, входящих только в одну из двух последовательностей
set_union	Создание отсортированного объединения множеств
make_heap	Преобразование последовательности с произвольным доступом в пирамиду
pop_heap	Извлечение элемента из пирамиды
push_heap	Добавление элемента в пирамиду
sort_heap	Сортировка пирамиды

Домашнее задание

- Изучить оставшиеся алгоритмы самостоятельно