Стандартная библиотека шаблонов

Алгоритмы

Соглашение STL

- Функции, не являющиеся элементами классов, но предназначенные для работы с контейнерами.
- Описаны в заголовке <algorithm>. Некоторые численные алгоритмы — в <numeric>.
- В <algorithm> также описаны вспомогательные функции min(), max() и swap() — для двух значений произвольного типа.

Соглашение STL

- Все алгоритмы отделены от деталей реализации структур данных и используют в качестве параметров типы итераторов. Поэтому они могут работать с определяемыми пользователем структурами данных, когда эти структуры данных имеют типы итераторов, удовлетворяющие предположениям в алгоритмах.

Правила именования

- Для некоторых алгоритмов предусмотрены и оперативные и копирующие версии
- Когда копирующая версия предусмотрена для какого-то алгоритма *algorithm*, он называется *algorithm_copy*
- Алгоритмы, которые используют предикаты, оканчиваются суффиксом *_if*

Категории алгоритмов

- Не модифицирующие контейнер
- Модифицирующие контейнер
- Сортировки и подобные им операции
- Обобщенные числовые операции

Алгоритмы не модифицирующие контейнер

- Эти алгоритмы выполняются над каждым элементом диапазона, не изменяя их Например,

```
InputIterator find (InputIterator first,
                    InputIterator last, const T& value);
InputIterator find_if(InputIterator first,
                    InputIterator last, Predicate pred);
```

Алгоритмы не модифицирующие контейнер

- Например,

for_each(InputIterator first, InputIterator last,
Function f)

Другие алгоритмы

adjacent_find ()
count_if()
max_element()
search_n()
equal()
. . .

find ()
count()
min_element()
search()
mismatch()

Пример

```
int* location = find (numbers, numbers + 10,
                                        25);

bool div_3 (int a_) { return a_ % 3 ? 0 : 1; }
vector<int>::iterator iter = find_if (v.begin (),
                                    v.end (), div_3);
```

Пример

```
location = search (v1.begin (), v1.end (),
                   v2.begin (), v2.end ());
if (location == v1.end ())
cout << "v2 not contained in v1" << endl;
else
cout << "Found v2 in v1 at offset: "
        << location - v1.begin () << endl;
```

Модифицирующие контейнер алгоритмы

- Копировать

  copy() copy_backward()
- Обменять

  swap() iter_swap()

  swap_ranges()
- Преобразовать

  transform()
- Заменить

  replace() replace_if()

  replace_copy() replace_copy_if ()

Модифицирующие контейнер алгоритмы

- Заполнить

  fill() fill_n()
- Генерировать

  generate( )

  получает в качестве параметра функцию-генератор
- Удалить

  remove() remove_if()

  remove_copy() remove_copy_if()

Модифицирующие контейнер алгоритмы

- Убрать повторы

  unique() unique_copy()

Пример

```
int numbers[10];
generate (numbers, numbers + 10, rand);
```

Пример

```
class Fibonacci {
 public: Fibonacci () : v1 (0), v2 (1) {}
int operator () ();
private: int v1; int v2;
};
int Fibonacci::operator () () {
    int r = v1 + v2;
    v1 = v2; v2 = r;
    return v1;
}
```

Пример

```
int main () {
        vector <int> v1 (10);
        Fibonacci generator;
        generate (v1.begin (), v1.end (), generator);
        ostream_iterator<int> iter (cout, " ");
        copy (v1.begin (), v1.end (), iter);
        cout << endl;
        return 0;
}
```

Перестановки

- Расположить в обратном порядке

  reverse() reverse_copy()
- Переместить циклически

  rotate() rotate_copy()
- Перемешать

  random_shuffle
- Разделить

  partition() stable_partitoin()
- Лексикографическая перестановка

  next_permutation() prev_permutation()

Сортировки

- Все операции имеют две версии: одна берёт в качестве параметра функциональный объект типа *Compare*, а другая использует *operator<*
- Сортировка

  sort() stable_sort()

  partial_sort() partial_sort_copy()
- N-ный элемент

  nth_element()

Сортировки

- Бинарный поиск

  Для итераторов прямого доступа выполняются с линейной сложностью

  Для итераторов произвольного доступа выполняются с логарифмической сложностью.

  lower_bound() upper_bound()

  equal_range() binary_search()

- Слияние

  merge() inplace_merge()

Операции для множеств и пирамид

includes()
set_union()
set_intersection()
set_difference()
set_symmetric_difference()

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

Обобщенные численные операции

- Накопление

```
int sum = accumulate (v.begin (), v.end (), 0);


int mult (int initial_, int element_) {
return initial_ * element_;
}

int prod = accumulate (v.begin (), v.end (), 1,
                                        mult);
```

Обобщенные численные операции

- Скалярное произведение

```
result = inner_product (vector1, vector1 + 5,
                                    vector2, 0);


int add (int a_, int b_) { return a_ + b_; }
int mult (int a_, int b_) { return a_ * b_; }


int result = inner_product (v1.begin (), v1.end (),
   v2.begin (), 1 , add, mult);
```

Обобщенные численные операции

- Частичная сумма

partial_sum (v1.begin (), v1.end (), v2.begin ());

записывают во второй диапазон частичные суммы для каждой позиции в первом диапазоне:

*i1, *i1 + *i2, *i1 + *i2 + *i3

Обобщенные численные операции

- Смежные разности

adjacent_difference (v.begin (), v.end (),
result.begin ());

записывают во второй диапазон для каждой позиции разность её и предыдущего элемента:

*i1, *i2 - *i1, *i3 - *i2, ...