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 типов итераторов используются имена, которые указывают какой итератор требуется в алгоритме
- Для некоторых алгоритмов предусмотрены и оперативные и копирующие версии
- Когда такая 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, оставляя их неизменными
- Например,

int count (InputIterator first, InputIterator last, const T& val)

int count_if (InputIterator first, InputIterator last, UPredicate pred)

Алгоритмы не модифицирующие контейнер

```
InputIterator find (InputIterator first,
                    InputIterator last, const T& value);
InputIterator find_if(InputIterator first,
                    InputIterator last, UPredicate pred);
InputIterator find_if_not(InputIterator first,
                    InputIterator last, UPredicate pred);
bool equal ( InputIterator1 first1,
                    InputIterator1 last1,
                    InputIterator2 first2 )
```

Алгоритмы не модифицирующие контейнер

Function for_each(InputIterator first,
InputIterator last,
Function f)

Другие алгоритмы

find_first_of ()	find_end()
search()	search_n()
mismatch()	all_of()
any_of()	none_of()

. . .

Пример

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

vector <int> v, v1, v2;
bool div_3 (int a_) { return a_ % 3 ? 0 : 1; }
vector<int>::iterator iter;
iter = find_if (v.begin (), v.end (), div_3);
```

Пример

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

Модифицирующие контейнер алгоритмы

- Копировать

OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);

OutputIterator copy_n (InputIterator first, Size n, OutputIterator result);

OutputIterator copy_if (InputIterator first, InputIterator last, OutputIterator result, UPredicate pred);

BidirectionalIterator2

copy_backward (BidirectionalIterator1 first,
BidirectionalIterator1 last,
BidirectionalIterator2 result)

Модифицирующие контейнер алгоритмы

- Обменять

  swap();

  void iter_swap (ForwardIterator1 a,
  ForwardIterator2 b) ;

ForwardIterator2 swap_ranges (ForwardIterator1 first1,
ForwardIterator1 last1,
ForwardIterator2 first2)

- Другие

transform()	replace()	replace_if()
replace_copy()	fill()	fill_n()

Модифицирующие контейнер алгоритмы

- Генерировать

  generate( )

  получает в качестве параметра функцию-генератор

- Удалить

  remove() remove_if()

  remove_copy() remove_copy_if()

- Убрать повторы

  unique() unique_copy()

Пример

```
int numbers[10];
// function generator:
int RandomNumber () {
return std::rand()%100);
}
generate (numbers, numbers + 10,
                    RandomNumber);
```

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

void reverse (BidirectionalIterator first,
BidirectionalIterator last)

OutputIterator reverse_copy (BidirectionalIterator first,
BidirectionalIterator last, OutputIterator result)

- Переместить циклически

  rotate() rotate_copy()

- Перемешать

  random_shuffle()

- Разделить

  partition() stable_partitoin()

Сортировки

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

Сортировки

- Сортировка

```
void sort (RandomAccessIterator first,
                RandomAccessIterator last);
void sort (RandomAccessIterator first,
                RandomAccessIterator last,
                Compare comp);
bool is_sorted (ForwardIterator first,
                ForwardIterator last);
```

Сортировки

- Другие

stable_sort()

partial_sort()

partial_sort_copy()

- N-ный элемент

nth_element()

Сортировки

- Бинарный поиск

  Работают с итераторами произвольного доступа, уменьшая число сравнений, которое будет логарифмическим для всех типов итераторов. Для итераторов не произвольного доступа они выполняют линейное число шагов

lower_bound() upper_bound()

equal_range() binary_search()

- Слияние

merge() inplace_merge()

Дополнительная информация

http://www.cplusplus.com/reference/algorithm/

Обобщенные численные операции

- <numeric>

```
T accumulate (InputIterator first,
                    InputIterator last, T init);

T accumulate (InputIterator first,
                    InputIterator last, T init,
                     BinaryOperation binary_op);
```

Пример

- Сумма

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

- Произведение

```
int mult (int val, int elem) {
        return  val* elem;
}
int prod = accumulate (v.begin (), v.end (), 1,
                                    mult);
```

Пример

- Используем функциональный объект

```
int init = 100;
int numbers[] = {10,20,30};
 cout<<accumulate (numbers, numbers+3,
                   init, minus<int>());
```

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, mult, add);
```

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, ...

Класс string

- Шаблон

basic_string

- Классы

string

u16string

u32string

wstring

Конструкторы

```
string();
string (const string& str);
string (const string& str, size_t pos,
                size_t len = npos);
string (const char* s);
string (const char* s, size_t n);
string (size_t n, char c);
template <class InputIterator>
   string (InputIterator first, InputIterator last);
```

Операции

operator=

operator[ ]

operator+=

operator+

operator<<

operator>>

и все операции сравнения

Итераторы

begin ()

end ()

rbegin()

rend()

cbegin()

cend()

crbegin()

crend()

Методы для операций со строками

size()

max_size()

resize()

clear()

at()

back()

insert()

replace()

find()

. . .

length()

capacity()

reserve()

empty()

front()

erase()

copy()

compare()