

Технология ООП

Санкт-Петербургский государственный политехнический университет

11 октября 2011

Применяются ко всем контейнерным типам, а также к массивам встроенных типов. Эти алгоритмы связываются с определенным типом контейнера с помощью передачи им в качестве параметров пары соответствующих итераторов. Чтобы использовать алгоритмы нужно подключить файл `<algorithm>`.

Алгоритм `std::for_each`

Инкапсулирует цикл перебирающий элементы последовательности и вызывает заданный предикат для очередного элемента последовательности.

```
template<class IT, class PRED>
void for_each(IT first, IT last, PRED F)
{
    while(first != last) {
        F(*first);
        ++first;
    }
}

template<class InputIt, class UnaryFunction>
constexpr UnaryFunction for_each(InputIt first,
    InputIt last, UnaryFunction f)
{
    for (; first != last; ++first)
        f(*first);
    return f; // std::move since C++11
}
```

Пример задания предиката глобальной функцией

Задание: вывести куб каждого элемента последовательности.

```
int main()
{
    vector<int> v; //заполнение вектора
    for_each(v.begin(), v.end(), PrintCube); //

    //Поменять знак координат
    for_each(l1.begin(), l1.end(), Neg); //

    //Поменять знак координат через шаблон
    for_each(l1.begin(), l1.end(), Neg<int>); //
}
```

Алгоритмы поиска в контейнере

Используются в двух вариантах: `find()` — для поиска элемента по значению, `find_if()` — для поиска элемента, соответствующего некоторому условию.

```
template<typename IT, typename PRED> IT find_if(IT
    first, IT last, PRED f)
{
    while(first != last) {
        if(f(*first) == true){
            return first;
        }
        ++first;
    }
    return last;
}
```

Пример использования std::find

Задание: найти элемент последовательности, значение которого равно значению параметра используя find.

```
vector<Point> v; //формируются значения vector  
vector<Point>::iterator it = //FIXME
```

```
//Что должно быть определено в классе Point?  
if (/* ??? */)  
    cout << "Not found";
```

Пример использования `std::find_if` через глобальную функцию

Задание: найти все точки (`Point`), которые находятся в окружности с радиусом `== 10` от начала координат.

```
bool FindPoint(const Point& pt) {  
    int tmp = pt.m_x * pt.m_x + pt.m_y * pt.m_y;  
    return tmp * tmp < 10 * 10;  
}  
int main()  
{  
    vector<Point> v;  
    vector<vector<Point>::iterator> vIT;  
    vector<Point>::iterator it = v.begin();  
    while(it != v.end()) {  
        it = find_if(it, v.end(), FindPoint);  
        if(it != v.end())  
            vIT.push(it);  
    }  
}
```

Алгоритмы подсчета количества совпадений

Используются в двух вариантах: `count()` — для поиска элемента по значению, `count_if()` — для поиска элемента, соответствующего некоторому условию.

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

```
template<typename IT, typename PRED> int
count_if(IT first, IT last, PRED f)
{
    int count = 0;
    while(first != last) {
        if(f(*first) == true) {
            count++;
        }
        ++first;
    }
    return count;
}
```

Пример использования `std::count_if` через функциональный объект

Преимущество функционального объекта — можно запаковать любое количество информации.

Задание: посчитать точки, которые отстоят меньше, чем на заданное значение от начала координат.

```
class COUNT_IF {  
    int m_d;  
    public :  
    COUNT_IF(int d){ m_d = d; }  
    bool operator()(const Point& pt) { ...  
        return tmp * tmp < m_d * m_d;  
    }  
};  
  
size_t n = count_if(v.begin(), v.end(), COUNT_IF(2)  
    );
```

Алгоритмы копирования последовательностей

Копируют все элементы в диапазоне итераторов (`std::copy`) и элементы соответствующие условию (`std::copy_if`).

```
template<typename FROM, typename TO, typename PRED>
void copy_if(FROM first, FROM last, TO res, PRED f)
{
    while(first != last) {
        if(f(*first) == true) {
            *res = *first; ++res;
        }
        ++first;
    }
}
```

Пример использования std::copy_if через шаблон класса

Задание: скопировать все элементы, кроме совпадающих с указанным значением.

```
template<typename T> class COPY_IF {  
    T m_t;  
    public:  
    COPY_IF(const T& t){ m_t = t; }  
    bool operator()(const T& t) { return t != m_t;  
    }  
};
```

```
vector<int> v1;  
int ar[] = {1, 5, -6, 1};  
//скопировать все значения, кроме 1  
copy_if(ar, ar+sizeof(ar)/sizeof(int), ??? ,  
        COPY_IF<int>(1));
```

```
//вывести на печать, кроме Point(1,1)  
copy_if(v.begin(), v.end(), ostream_iterator<Point  
>(cout, "\n"), COPY_IF<Point>(Point(1,1)));
```

Пример использования `std::copy_if()` через λ -выражения

Задание: скопировать все элементы, кроме совпадающих с указанным значением.

```
vector<Point> v; //формируем значения vector  
vector<Point> v1; //пустой вектор!!!  
Point point(1, 1);  
  
copy_if(v.begin(), v.end(), back_inserter(v1),  
        [point] (const Point &p) {  
            return p != point; }  
);  
  
copy_if(v.begin(), v.end(), back_inserter(v1),  
        [point] (const auto &p) {  
            return p != point; }  
); //Generic lambda (C++14)
```

Пример использования `copy_if()`

Чтение из файла значений через потоковый итератор не содержащих 33.

```
vector<int> v1;  
ifstream file("test.txt");  
copy_if(istream_iterator<int> (file),  
        istream_iterator<int>(),  
        inserter(v1, v1.begin()),  
        COPY_IF<int>(33));
```

Алгоритм сортировки `std::sort`

Использует функцию `quicksort` и начиная с определенных версий компилятора `introsort` (`quicksort` + `heapsort`).

Существуют два перегруженных варианта:

- без предиката через `operator<` сортирует последовательность в порядке возрастания значений;
- с указанием предиката.

Замечание: для класса **`list`** сортировка реализована **методом** класса.

Пример использования std::sort

```
int ar[5] = {-5, 8, 1, -10, 5};  
sort(ar, ar+5);  
vector<int> v(ar, ar+5);  
sort(v.begin(), v.end()); //в возрастающем порядке  
sort(v.rbegin(), v.rend()); //в убывающем порядке
```

Пример использования сортировки с предикатом

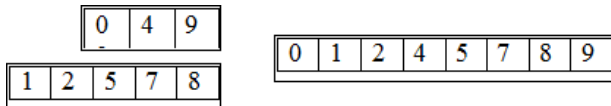
```
vector<Point> v;  
//Отсортировать по удалению от x, y  
class CMP_IF {  
    int x, y;  
    public:  
    CMP_IF(int a, int b) { x = a; y = b; }  
    bool operator()(const Point& pt1, const Point&  
        pt2) {  
        return FindDist(pt1) < FindDist(pt2);  
    }  
    int FindDist(const Point& pt);  
};  
sort(v.begin(), v.end(), CMP_IF(2, 2));
```

Объединение последовательностей через алгоритм merge

Объединение двух **отсортированных** последовательностей.

```
template<class InputIt1, class InputIt2, class
    OutputIt>
OutputIt merge(InputIt1 first1, InputIt1 last1,
               InputIt2 first2, InputIt2 last2,
               OutputIt d_first) //Compare comp
{
    for (; first1 != last1; ++d_first) {
        ...
        if (*first2 < *first1) //comp(*first2, *
            first1) {
            *d_first = *first2; ++first2;
        }
        else {
            *d_first = *first1; ++first1;
        }
    }
    return std::copy(first2, last2, d_first);
}
```

Пример использования std::merge



Задание: объединить vector и массив в список.

```
vector<int> v; //формирование значений
int ar[5] = {1, 2, 5, 7, 8};
sort(      ); //FIXME
sort(      ); //FIXME
```

```
list<int> l;
merge(     ); //FIXME
```

Трансформация последовательностей

Алгоритм `std::transform` существует в двух вариантах:

```
// трансформация одной последовательностей в другую  
template<class FROM, class TO, class UnaryF> TO  
    transform(FROM First1 , I FROM Last1 , TO Result ,  
        UnaryF f );  
// трансформация двух последовательностей в третью  
template<class FROM1, class FROM2, class TO, class  
    BinaryF> TO transform(FROM1 First1 , FROM1 Last1  
        , FROM2 First2 , TO Result , BinaryF f );
```

Пример трансформации одной последовательности

Задание: требуется преобразовать последовательность

$$b[i] = -a[i]$$

```
int a[] = {2, -5, 7, ...};
```

```
int b[sizeof(a)];
```

```
transform(a, a+sizeof(a)/sizeof(int), b,  
          negate<int>());
```

transform выполняет только замещение элементов, а не добавление новых.

Пример трансформации последовательности с добавлением

Задание: требуется преобразовать одну последовательности в другую по формуле: $l[i] = A * v[i] * v[i] + B * v[i] + C$

```
vector<double> v; //сформировали значения  
list<double> l; //пустой список
```

```
transform(v.begin(), v.end(), ???, [](double i) {  
    return A*i*i + B*i + C; } ); //FIXME
```

Пример трансформации двух последовательностей

Задание: требуется преобразовать две последовательности в третью по формуле: $d[i] = v[i] + l[i]$

```
vector<int> v; //сформировали значения  
list<int> l; //сформировали значения  
deque<int> d; //пустой контейнер  
  
transform(v.begin(), l.begin(), d.begin(), plus<int>()); //FIXME
```

Использование предикатов через метод класса

Существует проблема использования метода класса как предиката, т.к. в методе класса всегда присутствует неявно передаваемый указатель на текущий объект — «this». В STL предлагается шаблон функции `mem_fun_ref`, которая принимает в качестве параметра указатель на метод класса, а возвращает объект типа `mem_fun_ref_t`, в котором: сохраняется указатель на метод класса и перегружается `operator()`. Используя данную особенность становится возможным вызывать метод класса как предикат, через создание промежуточного объекта.

- `mem_fun` используется для последовательностей, в которых хранятся указатели;
- `mem_fun_ref` — для последовательностей, в которых хранятся копии объектов.

Пример использования std::mem_fun

```
class Point {  
    int m_x, m_y;  
    public:  
    Point (int x = 0, int y = 0) { m_x = x; m_y = y  
        ; }  
    bool MEM_F() { return ( m_x < 0 || m_y < 0 ); }  
};
```

```
Point arpt [] = { Point(), Point(1, 2), Point(3, 4)  
    };  
n = count_if(arpt, arpt+sizeof(arpt)/sizeof(Point),  
    mem_fun_ref(&Point::MEM_F)  
);
```

```
Point* arptr [] = { new Point(), new Point(1, 2),  
    new Point(3, 4) };  
int n = count_if(arptr, arptr+sizeof(arptr)/sizeof(  
    Point*), mem_fun(&Point::MEM_F));
```