

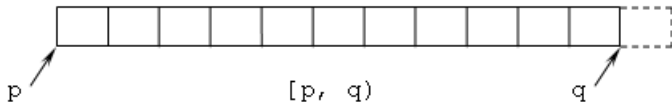
Семестр 2. Лекция 5. Алгоритмы.

Евгений Линский

24 Марта 2017

“Вспомнить все”

- ▶ Итераторы предназначены для перебора элементов контейнеров
 - `std::vector<int>::iterator`
 - `std::list<double>::iterator`
- ▶ Их универсальность позволяет писать код, не зависящий от типа контейнера (например, алгоритмы в STL)
- ▶ Итераторы имеют семантику и синтаксис указателей:
 - Поддерживают операции `++`, `*`, `->`
 - Последовательность задается двумя итераторами



Функторы — это классы, объекты которых похожи на функцию, в них перегружен оператор `()`.

```
struct sum_sq {  
    int operator()(int a, int b) const {  
        return a * a + b * b;  
    }  
};  
  
sum_sq f;  
int res = f(3, 4);
```

Похоже по синтаксису на функции, но это все-таки класс ("внутри" можно хранить данные).

```
struct cmp {
    int value;
    cmp(int v) : value(v) {}

    bool operator()(int a) const {
        return a < value;
    }
};

cmp f(2);
bool b = f(13);
```

Если функтор возвращает bool, то его называют предикатом.

Еще пример.

```
struct accum {
    int acc;
    accum() : acc(0) {}

    void operator()(int a) {
        acc += a;
    }
};

accum f;
f(13);
f(16);
cout << f.acc; // 29
```

- ▶ В STL более 100 алгоритмов.
- ▶ Мы рассмотрим только некоторые примеры.
 - Микро-алгоритмы
 - Алгоритмы, не модифицирующие последовательности
 - Алгоритмы типа `find`
 - Модифицирующие алгоритмы

- ▶ `swap(T &a, T &b)`
- ▶ `iter_swap(It p, It q)` Меняет местами значения элементов, на которые указывают итераторы.
- ▶ `max(const T &a, const T &b)`
- ▶ `min(const T &a, const T &b)`

Пример использования предиката

- ▶ У *max* и *min* алгоритмов есть версии с тремя параметрами.
- ▶ Третий параметр принимает бинарный предикат, задающий упорядоченность объектов.

```
struct Person {
    int age;
    string name;
    string city;
    Person(...) {...}
};

struct by_city {
    bool operator()(const Person& p1,
                    const Person& p2) const {
        return p1.city < p2.city;
    }
};

Person p1(30, "V", "Msc");  Person p2(15, "K", "Spb");
by_city f;
std::max<Person, by_city> (p1, p2, f);
std::max(p1, p2, by_city());
```


- ▶ `size_t count(It p, It q, const T &x)` Возвращает, сколько раз элемент со значением `x` входит в последовательность, заданную итераторами `p` и `q`.
- ▶ `size_t count_if(It p, It q, Pr pred)` Возвращает, сколько раз предикат `pred` возвращает значение `true`.

```
vector<int> v;  
v.push_back(...);  
vector<int>::iterator p = v.begin();  
vector<int>::iterator q = v.end();  
count_if(p, q, divides_by(8)); //how many
```

- ▶ `find(It p, It q, const T &x)` Возвращает итератор на первое вхождение элемента `x` в последовательность, заданную итераторами `p` и `q`.
- ▶ `find_if(It p, It q, Pr pred)` Возвращает итератор на первый элемент, для которого предикат `pred` вернул значение `true`.
- ▶ `min_element(It p, It q)`
- ▶ `max_element(It p, It q)`

Алгоритмы типа find

- ▶ `equal(It p, It q, Itr i)` Сравнивает две последовательности на эквивалентность. Вторая последовательность задается одним итератором, так как последовательности должны быть одинаковой длины. Если вторая короче, то `undefined behaviour`.
- ▶ `pair<It, Itr> mismatch(It p, It q, Itr i)` Возвращает пару итераторов, указывающую на первое несовпадение последовательностей.
- ▶ `F for_each(It p, It q, F func)` Для каждого элемента последовательности применяет функтор `func`. Возвращает функтор `func` после его применения ко всем элементам.

```
accum a;  
a = for_each(v.begin(), v.end(), a()); // Sum all  
elements
```

- ▶ `fill(lt p, lt q, const T &x)` Заполняют последовательность значениями, равными значению `x`.
- ▶ `generate(lt p, lt q, F gen)` Заполняют последовательность значениями, сгенерированными функтором `gen` (например, генератором случайных чисел).
- ▶ `copy(lt p, lt q, ltr out)` Копирует в `out`
- ▶ `reverse(lt p, lt q)`
- ▶ `sort(lt p, lt q)`
- ▶ `transform(lt p, lt q, ltr out, F func)` К каждому элементу входящей последовательности применяет функтор `func` и записывает результат в последовательность, начинающуюся с итератора `out`.

```
template<class InputIt, class OutputIt>
OutputIt copy(InputIt first, InputIt last,
              OutputIt d_first) {

    while (first != last) {
        *d_first++ = *first++;
    }
    return d_first;
}

vector<int> v;
list<int> l;
copy(v.begin(), v.end(), l.begin());
```

```
template<class InputIt, class OutputIt,
         class UnaryPredicate>
OutputIt copy_if(InputIt first, InputIt last,
                OutputIt d_first,
                UnaryPredicate pred) {
    while (first != last) {
        if (pred(*first))
            *d_first++ = *first;
        first++;
    }
    return d_first;
}
copy(v.begin(), v.end(), l.begin(), divide_by(8));
```

Как записать тип pivot?

```
template<class It>
void q_sort(It p, It q) {
    ??? pivot = *p;
}
```

```
template <class T>
class vector {
    T *array;
    class iterator {
        typedef T value_type;
    };
};
```

```
template<class It>
void q_sort(It p, It q) {
    It::value_type pivot = *p;
}
```

Есть еще `iterator::pointer`, `iterator::reference`, `iterator::iterator_category`

- ▶ Random access iterator (RA)
Самый сильный итератор: поддерживает ++, --, арифметические операции типа +=.
- ▶ Bidirectional iterator (BiDi) Поддерживает только ++, --. Это более слабый итератор.
- ▶ Forward iterator (Fwd) (обсудим в другой раз) Поддерживает только ++.

У `std::vector` и `std::deque` RA итераторы, у остальных контейнеров - BiDi.

iterator_category

```
template <class T>
class vector {
    T *array;
    class iterator {
        typedef ra_tag iterator_category;
    };
};
```

```
template <class T>
class vector {
    Node *head;
    class iterator {
        typedef bidi_tag iterator_category;
    };
};
```

- ▶ `std::advance(it, n)` Продвигает итератор на `n` позиций вперед (аналогично `p += n` для указателей).
Для RA итераторов использует `+=`, для BiDi - `++`.
- ▶ `std::distance(it1, it2)` Возвращает расстояние между итераторами.

```
template <class Iterator>
Iterator advance(Iterator it, int amount)
{
    typedef Iterator::iterator_category tag;
    advance(it, amount, tag());
}
```

```
template <class RAIterator>
RAIterator advance(RAIterator it, int amount, ra_tag t) {
    return it + amount;
}

template <class BiDiIterator >
BiDiIterator advance(BiDi it, int amount, bidi_tag t)
{
    for (;amount; --amount) ++it;
    return it;
}
```