Конспект по C++

Черепанов Валерий

25 марта 2016 г.

Часть I
Лекция 0

1 Итераторы внутри STL

```
template<class Iter>
void sort(Iter p, Iter q);

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

Проблемы

1. Знаем итератор, но не знаем, например, тип элементов вектора.
2. Не знаем, что умеет итератор (например, может ли он в random access?). Поэтому большинство операций с итератором обарачиваем в библиотечные функции

   ```
   advance(Iter& it, int n);
   distance(Iter& it1, Iter& it2);
   ```

Решения

Как решена проблемы в STL?

```
template<class T>
class vector {
    T *array;
    class Iterator {
        typedef value_type T; // Решение первой проблемы
        // В sort пишем typename Iter::value_type var;
        typedef interator_category ra_iterator; // Решение второй проблемы
    };
};
```

Как делать “if” по типу? Перегрузкой!

```
template<class Iter>
void advance (Iter it, int n) {
    typename Iter::iterator_category ite;
    advance_impl(it, n, ine);
}
template<class Iter>
void advance_impl(Iter& it, int n, ra_iterator it) {
    it += n;
}
template <class Iter>
void advance_impl(Iter int n, int n, bidi_iterator it) {
    int i = 0;
    if (n > 0) {
        while(i < n) {
            ++it;
            ++i;
        }
    }
    if (n < 0) {
        while(i > n) {
            --it;
            --i;
        }
    }
}
```

Используется полиморфизм времени компиляции.

2 Iterator traits

Хотим делать примерно то же самое, но не для итераторов, а для указателей. Проблема:

```
template<class Iter>
void sort(Iter p, Iter q) {
    Iter::value_type;
}
```

Если вызовем sort от двух указателей, то получим compilation error.

Решение проблемы:

```
template<class Iter>
class iter_traits {
    typedef value_type Iter::value_type;
    typedef iterator_category Iter::iterator_category;
}
vector<int>;
typename iter_traits<vector<int>::iterator>::value_type a;
```

Кажется, мы ничего на самом деле не решили, а просто написали какую-то чушь. Но на самом деле это не так, нужно лишь воспользоваться специализацией шаблонов!

```
template<typename Iter*> // специализация для указателей
class iter_type {
    typedef value_type Iter;
};
```

Iter:iterator_category → iter_traits<Iter>::iterator_category