Контейнеры, итераторы, алгоритмы

МГТУ им. Н.Э. Баумана

May 7, 2016

Фундаментальные понятия STL – стандартной библиотеки шаблонов

- Контейнеры – обобщенные классы, позволяющие хранить пользовательские данные в структуре, представляющей гарантии сложности определенных операций: поиска элемента, обращения по индексу, удаления элемента и т.д.
  - vector<T>, array<T, Len> – массив
  - list<T> – список
  - map<T, V> – ассоциативный массив
  - set<T> – коллекция уникальных элементов ...
- Алгоритмы – обобщенные функции преобразования пользовательских данных, хранящихся в контейнере: копирование в другой контейнер, поэлементное преобразование данных, поиск, сортировка, подсчет количества, извлечение уникальных элементов (см. <algorithm>):
  - sort
  - equal ...
- Итераторы – связующая сущность между контейнерами и алгоритмами. Итераторы представляют интерфейс для доступа к элементам контейнера в алгоритмах.

Зачем нужны итераторы

For each in list

```
1  for (List* l = head; l != nullptr; l = l->next)
2    doSmth(l);
```

For each in array

```
1  for (size_t i = 0; i < arraysize(arr); ++i)
2    doSmth(arr[i]);
```

For each in map

```
1  for (size_t i = 0; i < arraysize(keys); ++i)
2    doSmth(values[i]);
```

For each in set

???

Проблема

Для каждого контейнера приходится делать свой алгоритм

Итератор

Обобщение указателя

Единый интерфейс, позволяющий перечислять все элементы в контейнере вне зависимости от внутренней реализации контейнера.
Интерфейс:

- `operator++()` – переход к следующему элементу
- `operator--()` – переход к предыдущему элементу
- `operator*()` – обращение к элементу по значению
- `operator->()` – обращение к элементу по указателю
- `operator==()` – указание на тот же элемент двух итераторов
- `operator!=()` – указание итератором на другой элемент

Интерфейс контейнера

- `container.begin()` – возврат итератора на первый элемент
- `container.end()` – возврат итератора на элемент, следующий за последним элементом в контейнере
- существует `N = container.size()`, такой что `container.begin()+ N == container.end()`
- для любого 0 <= k < N: `container.begin()+ k` $\in$ container

for_each с итератором

vector

```
1  for(auto it = vector.begin(); it != vector.end(); ++it) {
2    doSmth(*it);
3  }
```

list

```
1  for(auto it = list.begin(); it != list.end(); ++it) {
2    doSmth(*it);
3  }
```

set

```
1  for(auto it = set.begin(); it != set.end(); ++it) {
2    doSmth(*it);
3  }
```

template

```
1  template <typename Container>
2  void for_each(const Container& container, void (*doSmth)(const typename
       Container::value_type&))
3  {
4    for(auto it = container.begin(); it != container.end(); ++it)
5      doSmth(*it);
6  }
```

Делаем итератор для вектора

```
1  struct IntVector {
2    int* mData;
3    size_t mSize;
4    IntVector(size_t size) : mSize(size) {
5      mData = new int[mSize];
6    }
7    ~IntVector() { delete[] mData; }
8
9    struct Iterator {
10     int* mData;
11     size_t mCount;
12     Iterator(int* data, size_t count) : mData(data), mCount(count) {}
13     Iterator& operator++() { mCount+=1; return *this; }
14     bool operator==(const Iterator& it) { return mCount == it.mCount; }
15     bool operator!=(const Iterator& it) { return !(*this == it); }
16     int& operator*() { return mData[mCount]; }
17     int* operator->() { return &(mData[mCount]); }
18   };
19
20   Iterator begin() { return Iterator(mData, 0); }
21   Iterator end() { return Iterator(mData, mSize); }
22 };
```

Проверим работу

```
1  void setCounterValue(int& i) {
2    static int counter = 0;
3    i = counter++;
4  }
5
6  int main() {
7    IntVector v(10);
8    std::for_each(v.begin(), v.end(), setCounterValue);
9
10   for(IntVector::Iterator it = v.begin(); it != v.end(); ++it) {
11     std::cout << *it << std::endl;
12   }
13 }
```

```
1  void f(IntVector& v) {
2    std::cout << std::count(v.begin(), v.end(), 5) << std::endl; // ERROR:
3    // error: no matching function for call to 'count'
4
5    // note: candidate template ignored: substitution failure [with
         _InputIterator = IntVector::Iterator, _Tp = int]: no type named '
         difference_type' in 'std::__1::iterator_traits<IntVector::Iterator
         >'
6    // count(_InputIterator __first, _InputIterator __last, const _Tp&
         __value_)
7  }
```

Добавим элементы стандартных итераторов

```
struct Iterator {
    typedef std::ptrdiff_t difference_type; //type of it1-it2
    typedef int value_type; // type of container value
    typedef int* pointer; // type of container value ptr
    typedef int& reference; // type of container value reference
    typedef size_t size_type; // type of container size
    typedef std::forward_iterator_tag iterator_category; // category of
        iterator

    // ...
};
```

```
void f(IntVector& v) {
  std::cout << std::count(v.begin(), v.end(), 5) << std::endl; // OK!
}
```

Типы стандартных итераторов

- Random access iterator (RA)
  `++`, `--`, `*`, `->`, арифметические операции
  `random_access_iterator_tag`
- Bidirectional iterator (BiDi)
  `++`, `--`, `*`, `->`
  `bidirectional_iterator_tag`
- Forward iterator (Fwd)
  `++`, `*`, `->`
  `forward_iterator_tag`
- Output iterator
  `++`, запись значения
  `output_iterator_tag`
- Input iterator
  `++`, чтение значения
  `input_iterator_tag`