Iterators & Traits (01.12.17)

STL

- Контейнеры (vector, deque, map, ...)
- Итераторы
- Алгоритмы (find, sort, ...)
- advance, ...

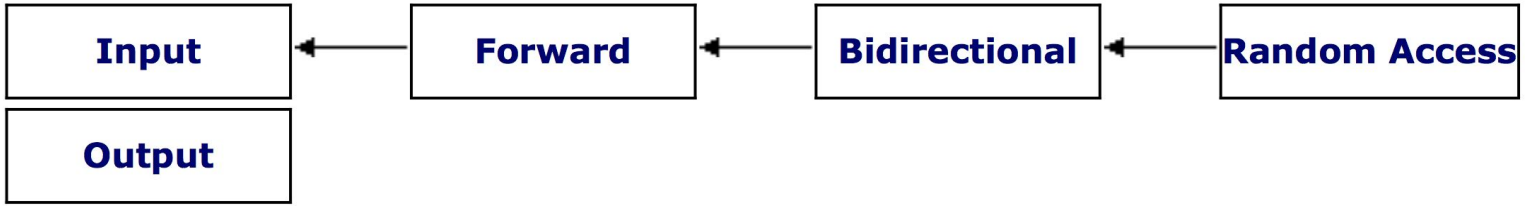
Итератор

сущность, которая

- указывает на некоторый элемент из некоторого диапазона
- может итерироваться (“пробегаться”) по элементам этого диапазона

Данным условиям удовлетворяют сырые указатели

Итераторы в STL



Все итераторы поддерживают:

- Инкремент (it++ и ++it)
- Копирование (it2 = it)

Итераторы в STL. Input, Output

Input:

- сравнение ==, !=
- разыменовывание как rvalue:
  - std::cout << *it; // OK
  - *it = 5; // WRONG

Output:

- разыменовывание как lvalue:
  *it = 5; // OK

Итераторы в STL. Input, Output

```
ifstream fin(“input.txt”);
vector<int> data;


std::copy(istream_iterator<int>(fin),
      istream_iterator<int>(), std::back_inserter(data));
std::copy(data.begin(), data.end(),
      ostream_iterator(cout))
```

Итераторы в STL. Forward

Forward =

- Input
- Output (если не константный)
- Multipass guarantee

Пример - итератор односвязного списка

Итераторы в STL. Bidirectional

Bidirectional =

- Forward
- Декремент (it-- и --it)

Пример - итератор двусвязного списка

Итераторы в STL. RandomAccess

RandomAccess =

- Bidirectional
- Сравнение <, >=, …
- Разыменование со смещением it[n]
- +, -
    - it + n
    - it2 - it

Пример - итератор вектора.

Итераторы в STL. Использование

- Работа с контейнерами
- Алгоритмы

```
std::count_if( vec.begin(), vec.end(),
    std::bind2nd( std::less<int>(), vec[0]  )  );
// кол-во элементов в векторе, которые меньше 0-ого
```

Traits. Advance

```
template<class IterT, class Distance>
void advance( IterT& it, Distance n );
```

Передвигает итератор на n позиций.

Но как она реализована?

Traits. Advance. Общая схема

```
template<class IterT, class Distance>
void advance( IterT& it, Distance n ) {
    if( it - RandomAccess it ) {
        it += n;
    } else {
        while(n--) ++it;
    }
}
```

Traits. Advance. Общая схема

Заведем структуру, которая будет содержать информацию об итераторе.

```
template<typename IterT> struct iterator_traits {

    // задает тип итератора

    typedef typename iterT::iterator_category  iterator_category;

    // задает тип значения

    typedef typename iterT::value_type  value_type;

    ...
```

Traits. Advance. Общая схема

```
template<class IterT, class Distance>
void advance( IterT& it, Distance n ) {
    if( typeid(typename std::iterator_traits<IterT>::iterator_category)
        == typeid(std::random_access_iterator_tag))
    {
        ...
```

Traits. Advance. Указатели

Внутрь указателей нельзя поместить дополнительную информацию, поэтому заведем специализацию шаблона

```
template<typename IterT> struct iterator_traits<IterT*> {
    // задает тип итератора
    typedef random_access_iterator_tag  iterator_category;
    ...
```

Traits. Advance. Недостатки

Так как *advance* - шаблонная функция, то ее инстанциация происходит на этапе компиляции ->

На этапе компиляции известен тип итератора. Поэтому если перенести проверку на этап компиляции, то

- ускорим время работы программы
- избавимся от ошибок компиляции

Как это сделать?

Traits. Advance. Перегрузка

```
template<class IterT, class Distance>
void advance( IterT& it, Distance n ) {
        doAdvance(
            it, d,
            typename std::iterator_traits<IterT>::iterator_category()
        );
}
```

Инвалидация итераторов

```
std::map<int, int> myMap;

...

for( auto it = myMap.begin(); it != myMap.end(); it++ ) {
        if( it->first == BadValue ) {
                myMap.erase( it );
        }
}
```