Итераторы в STL

Зачем нужны итераторы?

- Для унифицированного доступа к различным видам контейнеров. При этом удобно пользоваться алгоритмами из библиотеки <algorithm>
- Для безопасности при изменениях в контейнере + контроль выхода за пределы контейнера
- В некоторых структурах данных сложно определить, в каком порядке итерировать элементы. Для них может существовать несколько видов итераторов.

Пример кода без итераторов

```
std::vector<int> data;
…
for( int i = 0; i < data.size(); i++ ) {
        do_some_work( data[i] );
}
```

Теперь с итераторами

```
std::vector<int> data;
…
for( std::vector<int>::iterator it = data.begin(); it != data.end(); it++ ) {
       do_some_work( *it );
}
```

Или, с использованием шаблонов из <algorithm>

```
std::vector<int> data;
…
for_each( data.begin(), data.end(), do_some_work );
```

category				properties	valid expressions
all categories				*copy-constructible*, *copy-assignable* and *destructible*	X b(a); b = a;
				Can be incremented	++a a++
Random Access	Bidirectional	Forward	Input	Supports equality/inequality comparisons	a == b a != b
				Can be dereferenced as an *rvalue*	*a a->m
			Output	Can be dereferenced as an *lvalue* (only for *mutable iterator types*)	*a = t *a++ = t
				default-constructible	X a; X()
				Multi-pass: neither dereferencing nor incrementing affects dereferenceability	{ b=a; *a++; *b; }
				Can be decremented	--a a-- *a--
				Supports arithmetic operators + and -	a + n n + a a - n a - b
				Supports inequality comparisons (<, >, <= and >=) between iterators	a < b a > b a <= b a >= b
				Supports compound assignment operations += and -=	a += n a -= n
				Supports offset dereference operator ([])	a[n]

Класс iterator

iterator tag	Category of iterators
input_iterator_tag	Input Iterator
output_iterator_tag	Output Iterator
forward_iterator_tag	Forward Iterator
bidirectional_iterator_tag	Bidirectional Iterator
random_access_iterator_tag	Random-access Iterator

```
template<class _Category,
        class _Ty,
        class _Diff = ptrdiff_t,
        class _Pointer = _Ty *,
        class _Reference = _Ty&>
struct iterator
{// base type for iterator classes
        typedef _Category iterator_category;
        typedef _Ty value_type;
        typedef _Diff difference_type;
        typedef _Diff distance_type;// retained
        typedef _Pointer pointer;
        typedef _Reference reference;
};
```

Класс iterator_traits

```
template<class _Iter>
struct iterator_traits
{// get traits from iterator _Iter
        typedef typename _Iter::iterator_category iterator_category;
        typedef typename _Iter::value_type value_type;
        typedef typename _Iter::difference_type difference_type;
        typedef difference_type distance_type;// retained
        typedef typename _Iter::pointer pointer;
        typedef typename _Iter::reference reference;
};
```

Специальные виды итераторов

Predefined iterators
reverse_iterator
Reverse iterator (class template)
move_iterator
Move iterator (class template)
back_insert_iterator
Back insert iterator (class template)
front_insert_iterator
Front insert iterator (class template)
insert_iterator
Insert iterator (class template)
istream_iterator
Istream iterator (class template)
ostream_iterator
Ostream iterator (class template)
istreambuf_iterator
Input stream buffer iterator (class template)
ostreambuf_iterator
Output stream buffer iterator (class template)

Iterator generators:

back_inserter
Construct back insert iterator (function template)

front_inserter
Constructs front insert iterator (function template)

inserter
Construct insert iterator (function template)

make_move_iterator
Construct move iterator (function template)

Пример:

```
std::vector<int> data, data2;
…
std::back insert iterator< std::vector<int> > back_it( data2 );
std::copy( data.begin(), data.end(), back_it );
```

Безопасность (инвалидация) итераторов

- Основная проблема: как итератор узнает о том, что контейнер изменился?
- Решение: хранить в контейнере список всех итераторов и инвалидировать при любом потенциально опасном изменении.
- Чтобы это поддержать, в итератор добавляется указатель на следующий в списке (получается односвязный список итераторов)
- Для ускорения работы всё это можно выключить определяя символ #define NDEBUG