

# Язык C++

STL. Итераторы и основные алгоритмы

# Вычислительная сложность

- функцию зависимости объема работы, которая выполняется некоторым алгоритмом, от размера входных данных.
- Асимптотическая сложность ( $O(n)$ ,  $O(n*n)$ ,  $O(n*\log(n))$ )

# STL

- **Библиотека обобщенных компонент**
  - Контейнеры
  - Обобщенные алгоритмы
  - Итераторы
  - Функциональные объекты
  - Адаптеры
  - Аллокаторы
  - Вспомогательные функции
- **Гарантии производительности**

# Контейнеры

- Контейнеры последовательностей:
  - `vector<T>`
  - `deque<T>`
  - `list<T>`
  - `array<T>`
  - `forward_list<T>`
- Ассоциативные контейнеры:
  - `set<Key>` (`multiset`)
  - `map<Key, T>` (`multimap`)
- Неупорядоченные ассоциативные контейнеры
  - `unordered_set<Key>` (`multiset`)
  - `unordered_map<Ket, T>` (`multimap`)

# Обобщенные алгоритмы

- Find
- Max
- Merge
- Replace
- Sort
- ...

# Итераторы

- Указателеобразные объекты
- Связь между алгоритмами и контейнерами
- Категории
  - Выходные (*[LegacyOutputIterator](#)*)
  - Входные (*[LegacyInputIterator](#)*)
  - Однонаправленные (*[LegacyForwardIterator](#)*)
  - Двухнаправленные (*[LegacyBidirectionalIterator](#)*)
  - Произвольного доступа (*[LegacyRandomAccessIterator](#)*)
  - Непрерывный (C++17) ( *[LegacyContiguousIterator](#)*)
- Диапазон итераторов [first,last)
  - Корректный диапазон

# Входной итератор

```
template <typename InputIterator, typename T>
InputIterator find(InputIterator first,
                  InputIterator last,
                  const T& value)
{
    while (first != last && *first != value)
        ++first;
    return first;
}
```

# Входной итератор

Требования:

- `operator !=`
- `++iterator` и `iterator++`
- `value = *iterator`
- `operator ==`
- $O(1)$



# Выходной итератор

```
template <typename InputIterator, typename OutputIterator>
OutputIterator copy(InputIterator first,
                   InputIterator last,
                   OutputIterator result) {
    while (first != last) {
        *result = *first;
        ++first;
        ++result;
    }
    return result;
}
```

# Выходной итератор

Требования:

- `*iterator = value`
- `++iterator` и `iterator++`
- $O(1)$

# Однонаправленные итераторы

- Входной итератор
- Выходной итератор
- Сохранение для последующего использования

# Однонаправленные итераторы

```
template <typename ForwardIterator, typename T>
void replace(ForwardIterator first,
             ForwardIterator last,
             const T& x,
             const T& y) {
    while (first != last) {
        if (*first == x)
            *first = y;
        ++first;
    }
}
```

# Двунаправленные итераторы

```
#include <list>
#include <algorithm>

int main() {
    int a[10] = {12, 3, 25, 7, 11, 213, 7, 123, 29, -31};
    std::reverse(&a[0], &a[10]);

    std::list<int> l(&a[0], &a[10]);
    std::reverse(l.begin(), l.end());

    return 0;
}
```

# Двунаправленные итераторы

- Однонаправленный
- `operator--`

# Итераторы с произвольным доступом

```
vector<int> v;  
// .... Заполнение вектора  
bool b = binary_search(v.begin(),v.end(), 6);
```

# Итераторы с произвольным доступом

- Двухнаправленный итератор
- Достижение любой позиции за  $O(1)$
- Пусть  $r$  и  $s$  – итераторы с произвольным доступом,  $n$  – целое число, тогда:
  - $r+n$ ,  $n+r$ ,  $r-n$
  - $r[n]=*(r+n)$
  - $r+=n$ ,  $r-=n$
  - $r-s \rightarrow \text{int}$
  - $r<s$ ,  $r>s$ ,  $r\leq s$ ,  $r\geq s \rightarrow \text{bool}$



# Непрерывный итератор

- итератор произвольного доступа
- $*(a + n) = *(std::addressof(*a) + n)$



# Итераторы

- Описание контейнеров включает описание предоставляемых ими итераторов
- Описание обобщенных алгоритмов включает описание категорий итераторов с которыми они работают

Вывод:

*Интерфейсы контейнеров и алгоритмов STL спроектированы так, чтобы поддерживать эффективные комбинации и препятствовать неэффективным*

## iterator / const\_iterator

```
const vector<int> v(100,0);  
  
//vector<int>::iterator i = v.begin();    // !!Ошибка  
vector<int>::const_iterator i = v.begin();
```

# Итератор

Контейнер	Итератор	Тип
T a[n]	T*	Изм. Непрерывный
T a[n]	const T*	Конст. Непрерывный
vector<T>	vector<T>::iterator	Изм. Непрерывный
vector<T>	vector<T>::const_iterator	Конст. Непрерывный
deque<T>	deque<T>::iterator	Изм. Произв доступ
deque<T>	deque<T>::const_iterator	Конст. ,произв доступ
list<T>	list<T>::iterator	Изм., двунаправленный
list<T>	list<T>::const_iterator	Конст., двунаправленный

# Итераторы

Контейнер	Итератор	Тип
set<t>	set<t>::iterator	Конст., двунапр.
set<t>	set<t>::const_iterator	Конст., двунапр.
multiset<T>	multiset<T>::iterator	Конст., двунапр.
multiset<T>	multiset<T>::const_iterator	Конст., двунапр.
map<Key,T>	map<Key,T>::iterator	Изм., двунаправленный
map<Key,T>	map<Key,T>::const_iterator	Конст., двунапр.
multimap<Key,T>	multimap<Key,T>::iterator	Изм., двунаправленный
multimap<Key,T>	multimap<Key,T>::const_iterator	Конст., двунапр.

# Итераторы

Контейнер	Итератор	Тип
<code>unordered_set&lt;t&gt;</code>	<code>unordered_set&lt;t&gt;::iterator</code>	изм., однонапр.
<code>unordered_set&lt;t&gt;</code>	<code>unordered_set&lt;t&gt;::const_iterator</code>	Конст., однонапр.
<code>unordered_map&lt;Key,T&gt;</code>	<code>unordered_map&lt;Key,T&gt;::iterator</code>	Изм., однонапр.
<code>unordered_map&lt;Key,T&gt;</code>	<code>unordered_map&lt;Key,T&gt;::const_iterator</code>	Конст., однонапр.
<code>unordered_multiset&lt;t&gt;</code>	<code>unordered_multiset&lt;t&gt;::iterator</code>	изм., однонапр.
<code>unordered_multiset&lt;t&gt;</code>	<code>unordered_multiset&lt;t&gt;::const_iterator</code>	Конст., однонапр.
<code>unordered_multimap&lt;Key,T&gt;</code>	<code>unordered_multimap&lt;Key,T&gt;::iterator</code>	Изм., однонапр.
<code>unordered_multimap&lt;Key,T&gt;</code>	<code>unordered_multimap&lt;Key,T&gt;::const_iterator</code>	Конст., однонапр.

# Обобщенные алгоритмы

- Неизменяющие алгоритмы
- Изменяющие алгоритмы
- Связанные с сортировкой алгоритмы
- Обобщенные числовые алгоритмы



# Алгоритмы с предикатами

```
template<class Type>
struct greater {
    bool operator()( const Type& _Left, const Type& _Right ) const;
};

int main() {
    std::vector<int> v1;
    for (int i = 0 ; i < 8 ; i++ )
        v1.push_back( rand());

    std::sort( v1.begin(), v1.end(), greater<int>());
}
```

# Неизменяющие алгоритмы

- find
- adjacent\_find
- count
- for\_each
- mismatch
- equal
- search

# find и find\_if

```
class GreaterThan50 {
public:
    bool operator()(int x) const { return x > 50;}
};

int main() {
    std::vector<int> v;
    for (int i = 0; i < 13; ++i)
        v.push_back(i * i);

    std::vector<int>::iterator where;
    where = find_if(v.begin(), v.end(), GreaterThan50());

    assert(*where == 64);
    return 0;
}
```

# count

Задача: поиск количества значений равных данному

Сложность: линейная

```
int main() {  
    std::vector<int> v {0, 0, 1, 1, 1, 2, 2, 2};  
    std::cout << count(v.begin(), v.end(), 1) << std::endl;  
  
    int arr[] {1, 2, 3, 4, 5, 6, 7, 8};  
    std::cout << std::count_if(arr, arr + 8, even{}) << std::endl;  
  
    return 0;  
}
```

# Изменяющие алгоритмы

- copy
- copy\_backward
- fill
- generate
- partition
- random\_shuffle
- remove

- replace
- remove
- rotate
- swap
- swap\_ranges
- transform
- unique

# fill \ fill\_n

```
int main() {  
    std::vector<int> v1;  
    for (int i = 0 ; i <= 9 ; i++ )  
        v1.push_back( 5 * i );  
  
    fill(v1.begin( ) + 5, v1.end( ), 2);  
  
    fill_n( v1.begin( ) + 7, 3, 2 );  
}
```

# generate

Задача: Заполняет диапазон значениями генерируемыми подставленной функцией

Сложность: линейная

```
template <typename T>
class calc_square {
    T i;
public:
    calc_square(): i(0) {}
    T operator()() { ++i; return i * i; }
};

int main() {
    std::vector<int> v(10);
    std::generate(v.begin(), v.end(), calc_square<int>());
}
```

# erase\remove

Задача: удалить из диапазона элементы равные заданному

Сложность: линейная

```
int main() {  
    std::vector<int> vec = {1, 2, 0, 3, 4, 0, 5, 6, 7, 0, 8};  
  
    std::vector<int>::iterator new_end = std::remove(vec.begin(), vec.end(), 0);  
  
    vec.erase(new_end, vec.end());  
}
```



# Теоретико-множественные операции

- includes
- set\_union
- set\_intersection
- set\_difference
- set\_symmetric\_difference

# includes

Задача: Проверить содержится ли элементы одного сортированного диапазона в другом сортированном диапазоне

```
int main() {  
    bool result;  
  
    std::vector<char> v1 = to_vector("abcde");  
    std::vector<char> v2 = to_vector("aeiou");  
  
    result = std::includes(v1.begin(), v1.end(), v2.begin(), v2.end());  
    result = std::includes(v1.begin(), v1.end(), v2.begin(), v2.begin() + 2);  
  
    return 0;  
}
```

# set\_union

```
int main() {  
    std::vector<char> v1 = to_vector("abcde");  
    std::vector<char> v2 = to_vector("aeiou");  
  
    std::vector<char> setUnion;  
    std::set_union(  
        v1.begin(), v1.end(),  
        v2.begin(), v2.end(),  
        back_inserter(setUnion)  
    );  
  
    return 0;  
}
```

# set\_intersection / set\_difference / set\_symmetric\_difference

```
int main() {  
    std::vector<char> v1 = to_vector("abcde");  
    std::vector<char> v2 = to_vector("aeiou");  
  
    std::vector<char> setIntersection;  
    std::set_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(),  
        back_inserter(setIntersection));  
  
    std::vector<char> setDifference;  
    std::set_symmetric_difference(v1.begin(), v1.end(), v2.begin(), v2.end(),  
        back_inserter(setDifference));  
  
    return 0;  
}
```

# Обобщенные числовые алгоритмы

- accumulate
- partial\_sum
- adjacent\_difference
- inner\_product

# accumulate

```
int main() {  
    std::vector<int> v1, v2(20);  
    for (int i = 1; i < 21; i++)  
        v1.push_back(i);  
  
    int total = std::accumulate(v1.begin(), v1.end(), 0);  
    int ptotal = std::accumulate(v1.begin(), v1.end(), 1, std::multiplies<int>());  
}
```

# inner\_product

Задача: Получить скалярное произведение двух диапазонов

```
int main() {  
    int x1[5], x2[5];  
    for (int i = 0; i < 5; ++i) {  
        x1[i] = i + 1;  
        x2[i] = i + 2;  
    }  
  
    int result = std::inner_product(&x1[0], &x1[5], &x2[0], 0);  
  
    result = std::inner_product(&x1[0], &x1[5], &x2[0], 1, std::multiplies<int>(),  
std::plus<int>());  
}
```