

STL

Числовые пределы

Для того чтобы узнать числовые пределы в C++ используется шаблонный класс `numeric_limits`, который находится в заголовочном файле `<limits>`

```
int main()
{
    std::cout << "type\tlowest\thighest\n";
    std::cout << "int\t"
               << std::numeric_limits<int>::lowest() << '\t'
               << std::numeric_limits<int>::max() << '\n';
    std::cout << "float\t"
               << std::numeric_limits<float>::lowest() << '\t'
               << std::numeric_limits<float>::max() << '\n';
    std::cout << "double\t"
               << std::numeric_limits<double>::lowest() << '\t'
               << std::numeric_limits<double>::max() << '\n';
}
```

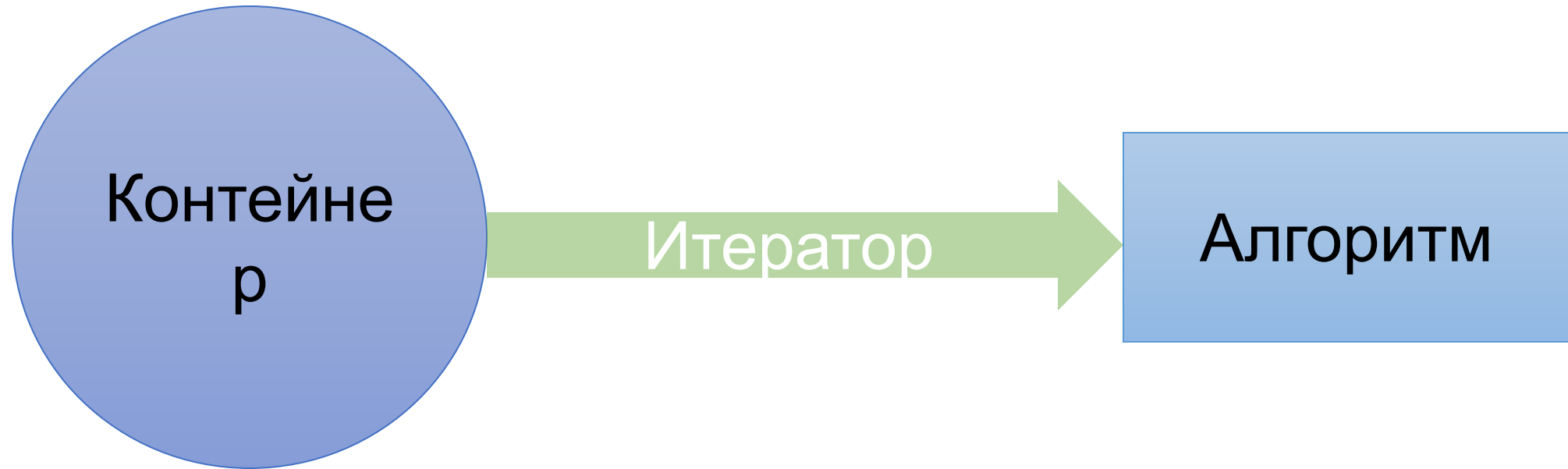
public-поля и методы numeric_limits

http://en.cppreference.com/w/cpp/types/numeric_limits

Компоненты STL

- Контейнеры – предназначены для управления коллекциями объектов определенного типа. У каждой разновидности контейнера свои достоинства и недостатки.
- Итераторы – предназначены для перебора элементов в контейнерах. Предоставляют стандартный интерфейс для любого типа контейнера.
- Алгоритмы – предназначены для обработки коллекций. Например, поиск, сортировка, модификация и т.д.

Концепция STL



Требования к хранимым объектам

1. copy – constructable
2. assignable
3. «Стандартная семантика»

Категории контейнеров

- Последовательные
- Ассоциативные
- Неупорядоченные ассоциативные
- Адаптеры контейнеров

Общие методы контейнеров

1. Конструктор по умолчанию, конструктор копирования, оператор присваивания, деструктор
2. `begin()`, `end()`
3. Операторы сравнения: `==`, `!=`, `>`, `>=`, `<`, `<=`
4. `size()`
5. `empty()`
6. `swap(obj2)`

Общие типы контейнеров

1. `C::value_type`
2. `C::reference`
3. `C::const_reference`
4. `C::pointer`
5. `C::iterator`
6. `C::const_iterator`
7. `C::size_type`

Последовательные контейнеры

- Статический массив (array)
- Вектор (vector)
- Дека (deque)
- Односвязный список (forward_list)
- Двусвязный список (list)

Общие члены последовательных контейнеров

1. Конструктор от 2 итераторов
2. Конструктор от count и defVal
3. Двухитераторный erase
4. push_back, pop_back, back
5. front
6. assign от двух итераторов
7. assign от count и val
8. insert от итератора и val
9. insert от итератора, n и val
10. insert от трех итераторов

vector

C – подобный динамический массив произвольного доступа с автоматическим изменением размера при добавлении элементов

1. operator[], at
2. resize
3. capacity, reserve

<http://en.cppreference.com/w/cpp/container/vector>

array

Array – контейнер, являющийся оберткой над обычным массивом фиксированного размера.

<http://en.cppreference.com/w/cpp/container/array>

deque

<http://en.cppreference.com/w/cpp/container/deque>

forward_list / list

http://en.cppreference.com/w/cpp/container/forward_list

<http://en.cppreference.com/w/cpp/container/list>

Ассоциативные контейнеры

- set <http://en.cppreference.com/w/cpp/container/set>
- map <http://en.cppreference.com/w/cpp/container/map>
- multiset <http://en.cppreference.com/w/cpp/container/multiset>
- multimap <http://en.cppreference.com/w/cpp/container/multimap>

Неупорядоченные ассоциативные контейнеры

- unordered_set
http://en.cppreference.com/w/cpp/container/unordered_set
- unordered_map
http://en.cppreference.com/w/cpp/container/unordered_map
- unordered_multiset
http://en.cppreference.com/w/cpp/container/unordered_multiset
- unordered_multimap
http://en.cppreference.com/w/cpp/container/unordered_multimap

Контейнеры - адаптеры

- stack <http://en.cppreference.com/w/cpp/container/stack>
- queue <http://en.cppreference.com/w/cpp/container/queue>
- priority_queue
http://en.cppreference.com/w/cpp/container/priority_queue

Итераторы

<http://en.cppreference.com/w/cpp/iterator>

```
#include <vector>
#include <forward_list>
#include <iostream>

int main()
{
    std::vector<int> elements { 1, 2, 3, 4, 5 };
    //std::forward_list<int> elements { 1, 2, 3, 4, 5 };

    /*std::vector<int>::const_iterator*/ auto iter = elements.cbegin();

    for(; iter != elements.cend(); ++iter)
        std::cout << *iter << ' ';

    return 0;
}
```

Итераторы

```
#include <array>
#include <iostream>

int main()
{
    std::array <float> floats = { 1.f, 2.f, 0.f, -1.f, 2.f, 124.f, 2.1f };

    std::cout << "size: " << floats.size() << '\n';

    decltype (floats)::reverse_iterator insert_iter = floats.rbegin();

    while (insert_iter != floats.rend())
    {
        *insert_iter = -1;
        ++insert_iter;
    }

    for(auto each : floats)
        std::cout << each << ' ';
}
```

Алгоритмы

<http://en.cppreference.com/w/cpp/algorithm>

```
#include <vector>
#include <list>
#include <iostream>
#include <algorithm>
#include <iterator>

int main()
{
    srand(1024);
    std::vector<int> v(10);
    std::generate(v.begin(), v.end(), [] { return rand() % 100; });
    std::list<int> l;

    std::cout << "size: " << l.size() << std::endl;
    std::copy(v.begin(), v.end(), std::back_inserter(l));
    std::copy(l.cbegin(), l.cend(), std::ostream_iterator<int>(std::cout, " "));
    std::cout << '\n';
}
```

string

http://en.cppreference.com/w/cpp/string/basic_string