C++ Advanced

Стандартная библиотека шаблонов. Итераторы и алгоритмы

Автор курса

Кирилл Чернега

После урока обязательно

Повторите этот урок в видео формате на ITVDN.com

Проверьте как Вы усвоили данный материал на TestProvider.com

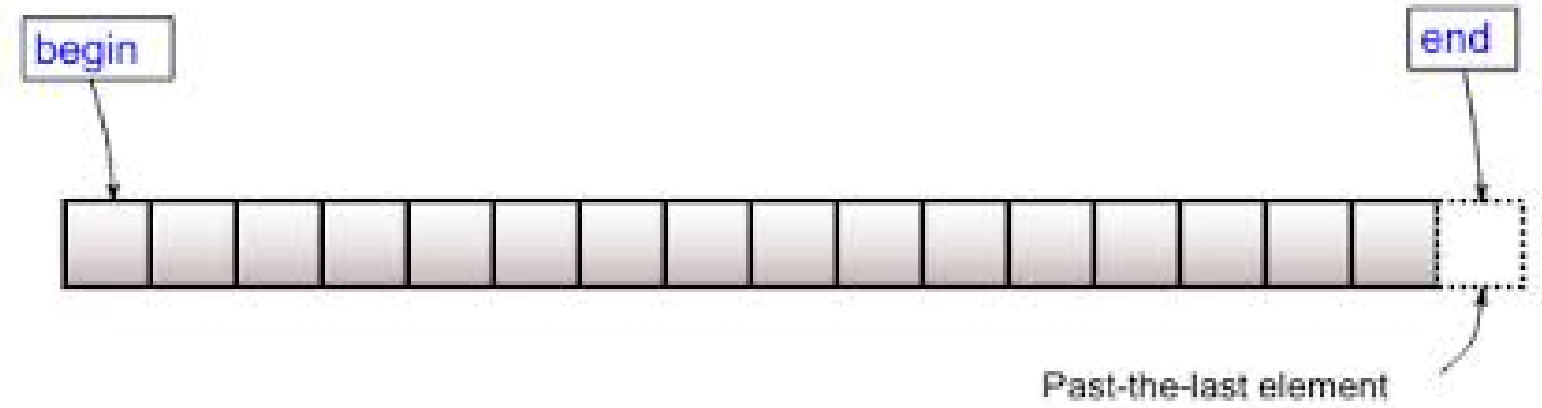
Стандартная библиотека шаблонов. Итераторы и алгоритмы.

Содержание урока

1. Типы итераторов.
2. Типы алгоритмов.
3. Использование алгоритмов.

Понятие итератора.

Итератор – это обобщение для указателя, умный указатель относительно контейнеров – обеспечивает стандартный интерфейс для доступа к элементам стандартных контейнеров либо своих собственных, которые поддерживают итераторы, без учета того, как устроен и разработан контейнер изнутри и какие данные хранятся внутри данного контейнера.



Типы итераторов.

Категории итераторов: Вывода, Ввода, Однонаправленные, Двунаправленные, Произвольного доступа

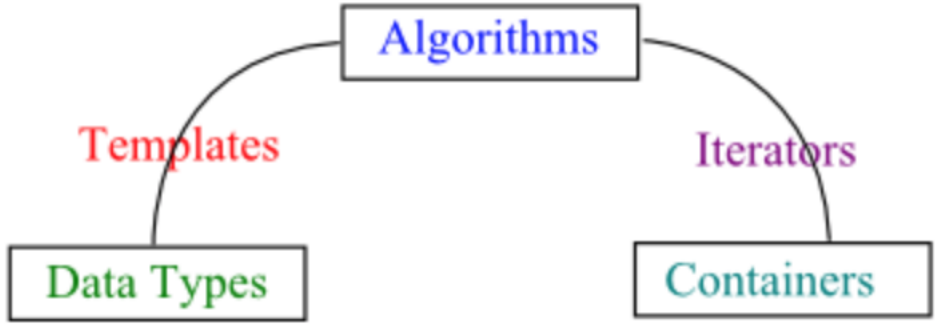
<table>
<tr><th colspan="4">Iterator category</th><th>Defined operations</th></tr>
<tr><td rowspan="4">RandomAccessIterator</td><td rowspan="3">BidirectionalIterator</td><td rowspan="2">ForwardIterator</td><td>InputIterator</td><td>• чтение
• инкремент (без нескольких проходов)</td></tr>
<tr><td>OutputIterator</td><td>• запись
• инкремент (без нескольких проходов)</td></tr>
<tr><td></td><td>• инкремент (с несколькими проходами)</td></tr>
<tr><td colspan="2"></td><td>• декремент</td></tr>
<tr><td colspan="3"></td><td>• случайный доступ</td></tr>
</table>

https://ru.cppreference.com/w/cpp/iterator

Типы итераторов. Продолжение.

Iterator Type	Behavioral Description	Operations Supported
random access (most powerful)	Store and retrieve values Move forward and backward Access values randomly	* = ++ –> == != — + – [ ] < > <= >= += –=
bidirectional	Store and retrieve values Move forward and backward	* = ++ –> == != —
forward	Store and retrieve values Move forward only	* = ++ –> == !=
input	Retrieve but not store values Move forward only	* = ++ –> == !=
output (least powerful)	Store but not retrieve values Move forward only	* = ++

Связь между алгоритмами, итераторами, контейнерами и типами данных.



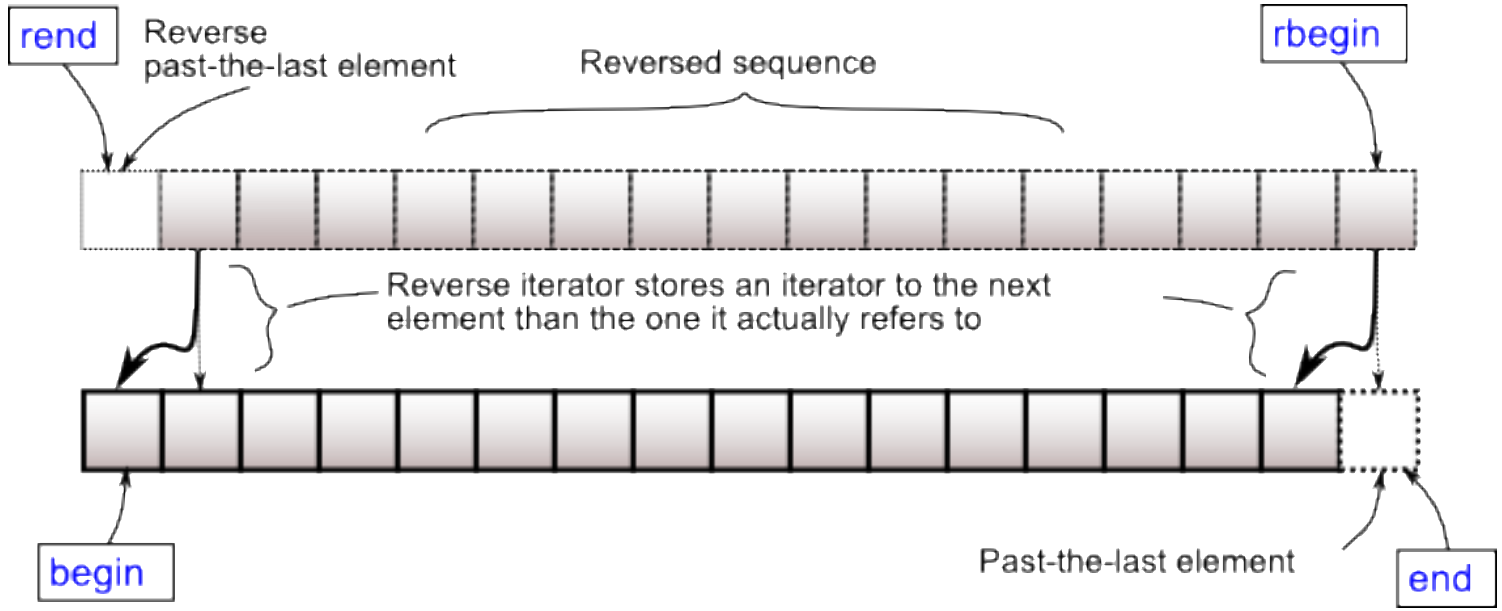
1. Templates
   make algorithms independent of the data types
2. Iterators
   make algorithms independent of the containters

https://github.com/caiorss/C-Cpp-Notes/blob/master/STL%20Iterators%20and%20Algorithms.org

Адапторы итераторов, операции над итераторами

std::advance() - смещение,
std::distance() – расстояние
(back_\front_\)inserter() –
вставка элементов в контейнер
по заданному итератору
next()\prev() - ++\--

std::move_iterator – проводит
смещение элементов, а не
копирование, в случае работы
с двумя контейнерами
(подробнее о move-семантике
смотрите в 7 уроке)



std::reverse_iterator

Бонус: const_iterator
std::_any_container_::(const_)iterator - _any_container_ - vector, deque, map, etc

Итераторы в контейнерах

Container Class	Iterator Type	Container Category
vector	random access	sequential
deque	random access	
list	bidirectional	
set	bidirectional	associative
multiset	bidirectional	
map	bidirectional	
multimap	bidirectional	
stack	none	adaptor
queue	none	
priority_queue	none	

Unordered – forward iterator

http://cs.stmarys.ca/~porter/csc/ref/stl/iterators.html

Алгоритмы STL

Не модифицирующие операции над последовательностями - *for_each*, *count_if*, *find_if*, etc

Модифицирующие операции над последовательностями – *copy*, *fill*, *transform*, *remove*, etc

Операции разделения - *partition* - *stable_partition*, *partition_copy*, etc

Операции сортировки (на отсортированных диапазонах) – *is_sorted*, *sort*, *stable_sort*, etc

Операции двоичного поиска (на отсортированных диапазонах) – *lower_bound*, *upper_bound*, etc

Операции над множествами (на отсортированных диапазонах) – *merge*, *set_intersection*, etc

Операции над кучей – *heap* - *make_heap*, *sort_heap*, *push_heap*, *pop_heap*, *is_heap*

Операции минимума/максимума – *min*, *max*, *min_element*, *max_element*, etc

Операции сравнения – *equal*, *lexicographical_compare*

Операции перестановки - *permutation* - *(is_\next_\prev_)permutation*

Числовые операции – *iota*, *accumulate*, *partial_sum*, etc

https://ru.cppreference.com/w/cpp/algorithm

Алгоритмы STL vs Самописных циклов

Преимущества алгоритмов STL:

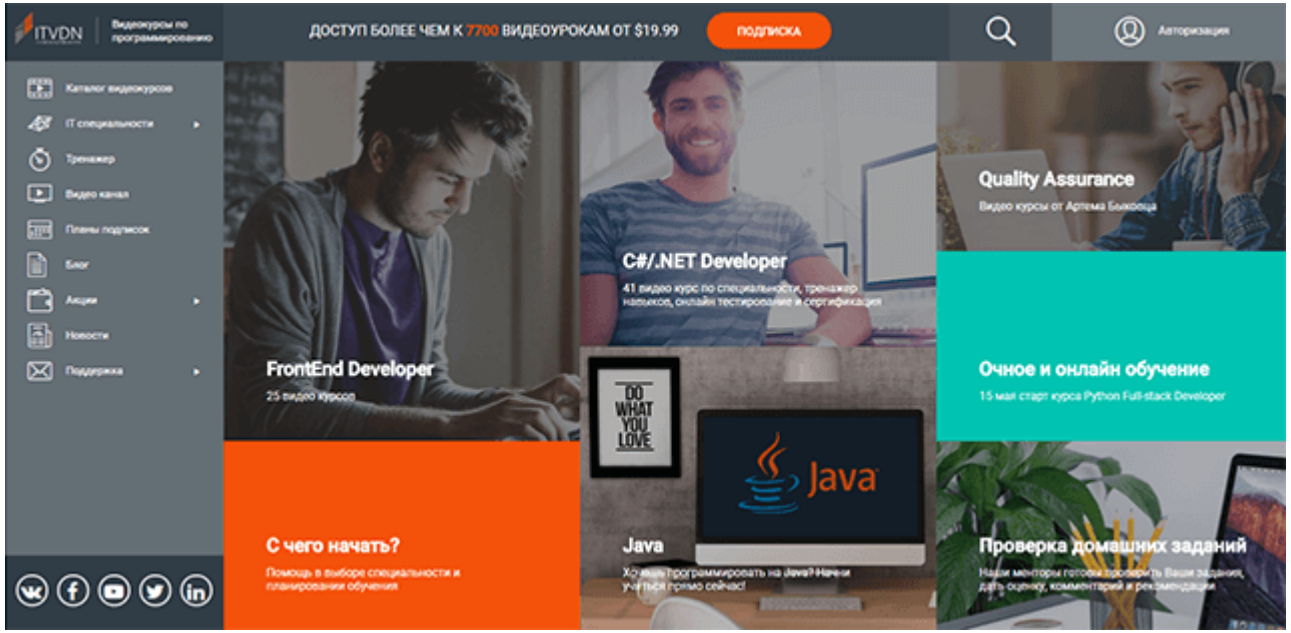
Ясность: по названию алгоритма понятно, что он делает, самодокументирующийся.
Правильность: шанс допустить ошибку при написании собственного «велосипеда» выше
Имплементация в STL безопасна.
Эффективность
STL имеет свои спецификации, оптимизации и прочие различные улучшения для различных стандартных алгоритмов, так как они пропитаны многолетним опытом Разработчиков. Так сказать, внутри их реализации содержатся best practices (но не всегда, зависит от специфики задачи, иногда для контроля лучше написать свое решение).

Резюме: в общем случае – преимущественно использовать стандартные алгоритмы из STL

https://www.codeproject.com/Articles/854127/Top-Beautiful-Cplusplus-std-Algorithms-Examples

Смотрите наши уроки в видео формате

ITVDN.com



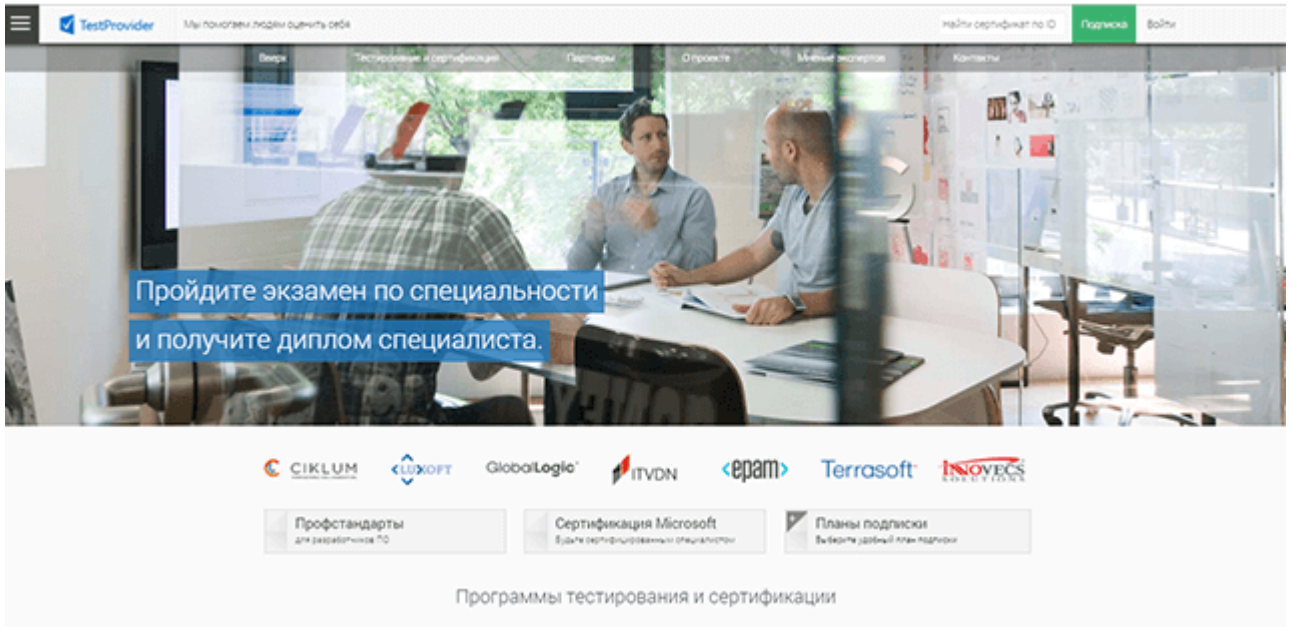
Посмотрите этот урок в видео формате на образовательном портале ITVDN.com для закрепления пройденного материала.

Курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics и другими высококвалифицированными разработчиками.

ITVDN
IT VIDEO DEVELOPERS NETWORK

Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

TestProvider

Q&A

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association