

Семинар #4: Контейнеры STL.

Часть 1: Контейнеры

Стандартная библиотека шаблонов (STL = Standard Template Library) включает в себя множество разных шаблонных контейнеров и алгоритмов для работы с ними.

контейнер	описание и основные свойства
<code>std::vector</code>	Динамический массив Все элементы лежат вплотную друг к другу, как в массиве Есть доступ по индексу за $O(1)$
<code>std::list</code>	Двусвязный список Вставка/удаление элементов за $O(1)$ если есть итератор на элемент
<code>std::forward_list</code>	Односвязный список Вставка/удаление элементов за $O(1)$ если есть итератор на предыдущий элемент
<code>std::set</code>	Реализация множества на основе сбалансированного дерева поиска Хранит элементы без дубликатов, в отсортированном виде Тип элементов должен реализовать <code>operator<</code> (или предоставить компаратор) Поиск/вставка/удаление элементов за $O(\log(N))$
<code>std::map</code>	Реализация словаря на основе сбалансированного дерева поиска Хранит пары ключ-значение без дубликатов ключей, в отсортированном виде Тип ключей должен реализовать <code>operator<</code> (или предоставить компаратор) Поиск/вставка/удаление элементов за $O(\log(N))$
<code>std::unordered_set</code>	Реализация множества на основе хеш-таблицы Хранит элементы без дубликатов, в произвольном порядке Поиск/вставка/удаление элементов за $O(1)$ в среднем
<code>std::unordered_map</code>	Реализация словаря на основе хеш-таблицы Хранит пары ключ-значение без дубликатов ключей, в произвольном порядке Поиск/вставка/удаление элементов за $O(1)$ в среднем
<code>std::multiset</code>	То же самое, что <code>std::set</code> , но может хранить дублированные значения
<code>std::deque</code>	Двухсторонняя очередь Добавление/удаление в начало и конец за $O(1)$
<code>std::stack</code> <code>std::queue</code> <code>std::priority_queue</code>	Стек Очередь Очередь с приоритетом
<code>std::pair</code>	Пара элементов, могут быть объектами разных типов Элементы пары хранятся в публичных полях <code>first</code> и <code>second</code>
<code>std::tuple</code>	Фиксированное количество элементов, могут быть объектами разных типов
<code>std::array</code>	Массив фиксированного размера, все элементы имеют один тип

Часть 2: Итераторы

Итератор – это абстракция для итерирования по контейнеру. Многие контейнеры STL имеют вложенный тип `iterator`. Объекты этого типа используются для итерирования по контейнеру. Контейнеры имеют следующие методы:

- `begin()` – возвращает итератор на первый элемент
- `end()` – возвращает итератор на фиктивный элемент, следующий после последнего

С итератором можно проводить следующие операции:

`*it`: перегруженный `operator*` – возвращает ссылку на элемент, на который указывает итератор

`it++`: переходим к следующему элементу

`it1 == it2` и `it1 != it2`: операторы равенства и неравенства

`it--`: переходим к предыдущему элементу (работает не для всех итераторов)

`it + n`: только для итераторов `vector` – переходим к элементу, следующему через `n` элементов после текущего.

`it1 - it2`: только для итераторов `vector` – возвращает расстояние между элементами

Рассмотрим пример работы с итераторами. Приведённый ниже код создаёт массив (`vector`) и множество на основе бинарного дерева, заполняет их элементами и печатает. Если для вектора поведение итератора очень похоже на указатель, то для множества оно сильно отличается. Например, перегруженный оператор `++` для итератора `set` – это нетривиальная операция перехода к следующему элементу дерева.

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;

int main () {
    vector<int> v = {54, 62, 12, 97, 41, 6, 73};
    for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;

    set<int> s = {54, 62, 12, 97, 41, 6, 73};
    for (set<int>::iterator it = s.begin(); it != s.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;
}
```

Задачи

- Напечатайте только чётные элементы вектора, используйте итераторы
- Напечатайте каждый второй элемент вектора, используйте итераторы
- Напишите функцию `inc`, которая будет принимать на вход вектор целых чисел типа `int` и будет увеличивать все элементы на 1. Для прохода по вектору используйте итераторы.
- Напечатать содержимое вектора в обратном порядке

Часть 3: `std::vector` и алгоритмы

В библиотеке `algorithm` содержится множество алгоритмов, предназначенных для работы с контейнерами STL.

`std::max_element` и `std::min_element`

Принимает на вход 2 итератора и возвращает итератор на максимальный элемент на подмассиве, задаваемом этими итераторами. Если таких элементов несколько, то возвращает итератор на первый из них.

Задачи:

- На вход подаётся n чисел. Напечатайте минимальный элемент и его индекс.

ВХОД	ВЫХОД
7	1 4
8 2 5 4 1 6 4	
1	1 0
1	

- На вход подаётся чётное количество чисел. Напечатайте минимальный элемент на первой половине и максимальный элемент второй половины.

ВХОД	ВЫХОД
8	2 9
7 2 8 4 1 9 4 2	
8	5 4
8 7 6 5 4 3 2 1	
2	5 1
5 1	

- На вход подаётся n чисел. Напечатайте максимальный элемент, который находится до минимального. Предполагается, что минимальный элемент не является первым.

ВХОД	ВЫХОД
7	8
7 2 8 4 1 9 4	
7	2
2 1 2 3 4 5 6	
2	3
3 1	

`std::find`

Принимает на вход 2 итератора и элемент того же типа, что и тип элементов вектора. Ищет этот элемент и возвращает итератор на этот элемент. Если этого итератора в контейнере нет, то возвращает итератор `end()` этого контейнера.

- На вход подаётся n чисел и ещё некоторое число. Напечатайте индекс этого числа в массиве. Если такого числа в массиве нет, то напечатайте `No such element`.

ВХОД	ВЫХОД
7	5
8 2 5 4 1 6 4	
6	
2	No such element
4 1	
5	

`std::sort`

Принимает на вход 2 итератора. Сортирует подмассив, задаваемый этими итераторами, по возрастанию. Сортировка работает за $O(n \log(n))$.

- На вход подаётся n чисел. Отсортируйте их и напечатайте.

ВХОД	ВЫХОД
8	1 2 4 4 5 6 8 9
8 2 5 4 9 1 6 4	

- На вход подаётся n строк. Отсортируйте их лексиграфически и напечатайте.

ВХОД	ВЫХОД
5	Cat Cattle Dog Dolphin Elephant
Cat Dog Elephant Cattle Dolphin	

`std::reverse`

Принимает на вход 2 итератора. Обращает подмассив, задаваемый этими итераторами.

- На вход подаётся n чисел. Найдите максимум и отсортируйте элементы, идущие до максимума по возрастанию, а все элементы, идущие после максимума – по убыванию.

ВХОД	ВЫХОД
8	2 4 5 8 9 6 4 1
8 2 5 4 9 1 6 4	

`std::count`

Принимает на вход 2 итератора и некоторое значение. Находит сколько элементов массива равны этому значению.

- На вход подаётся n чисел. Найдите сколько элементов массива равны максимальному.

ВХОД	ВЫХОД
8	3
8 2 5 8 8 1 6 4	

`std::accumulate` (библиотека `numeric`)

Принимает на вход 2 итератора и некоторый объект (начальное значение). Прибавляет все элементы из подмассива, задаваемого итераторами, к начальному значению. В итоге возвращает получившееся значение.

- На вход подаётся n чисел. Напечатайте сумму этих чисел.

ВХОД	ВЫХОД
8	39
8 2 5 4 9 1 6 4	
3	5000000000
2000000000 1000000000 2000000000	

- На вход подаётся n чисел и ещё целое число k . Напечатайте сумму k наименьших чисел.

ВХОД	ВЫХОД
8	7
8 2 5 4 9 1 6 4	
3	

Часть 4: std::pair

Пара – это простейший контейнер, который может хранить в себе 2 элемента (возможно, разных типов). Реализация пары имеет примерно следующий вид:

```
template <typename T1, typename T2> struct pair {
    T1 first;
    T2 second;
};
```

Для пары определены операторы сравнения. Сравнение происходит в лексиграфическом порядке. То есть для оператора больше сначала сравниваются первые элементы и только если они равны, сравниваются вторые.

Для простого создания пар есть шаблонная функция `make_pair`. Пару можно создать так:

```
std::pair<string, int> p1 = make_pair("Titanic", 8.4);
std::pair<string, int> p2 {"Titanic", 8.4};
```

- На вход подаётся n чисел. Отсортируйте их и напечатайте сами элементы и их старые индексы.

ВХОД	ВЫХОД
8	1 2 4 4 5 6 8 9
8 2 5 4 9 1 6 4	5 1 3 7 2 6 0 4

- На вход подаётся n фильмов. Передаются названия фильмов и их рейтинг на кинопоиске. Отсортировать эти фильмы по возрастанию рейтинга.

ВХОД	ВЫХОД
5	Venom2 6.2
TheMatrix 8.5	Shrek 8.0
Titanic 8.4	Titanic 8.4
GreenMile 8.9	TheMatrix 8.5
Shrek 8.0	GreenMile 8.9
Venom2 6.2	

Часть 5: `std::list`

`std::list` – Это двусвязный список. Основные методы для работы со списком:

метод	описание
<code>insert</code>	Принимает на вход итератор и некоторый объект и вставляет этот объект перед элементом, на который указывает итератор.
<code>erase</code>	Принимает на вход итератор и удаляет элемент. Переданный итератор, конечно, становится недействительным, поэтому этот метод возвращает корректный итератор на элемент, следующий за удалённым
<code>push_back</code> <code>push_front</code> <code>pop_back</code> <code>pop_front</code>	Добавить элемент в конец. Добавить элемент в начало. Удалить элемент из конца. Добавить элемент из начала.
<code>sort</code>	Сортирует список (функция <code>sort</code> из библиотеки <code>algorithm</code> для списка не работает)

К итераторам `std::list<T>::iterator` нельзя прибавлять целые числа, вместо этого нужно использовать функцию `std::advance`. Также эти итераторы нельзя вычитать, вместо этого нужно использовать функцию `std::distance`. Эти две функции работают за линейное время.

- На вход подаётся n чисел. Сохраните эти числа в связном списке, найдите их сумму и напечатайте её.
- На вход подаётся n чисел. Сохраните эти числа в связном списке, отсортируйте список и напечатайте их.
- На вход подаётся n чисел. Сохраните эти числа в связном списке. Скопируйте все элементы списка в его конец и напечатайте его.

ВХОД	ВЫХОД
5	8 2 1 4 2 8 2 1 4 2
8 2 1 4 2	

- Напишите функцию `insert_after`, которая будет принимать на вход список из чисел типа `int`, итератор на элемент этого списка и некоторое число `x`. Функция должна вставлять `x` после элемента, заданным итератором.
- Проверьте только что написанную функцию. На вход подаётся n чисел. Сохраните эти числа в связном списке. Продублируйте каждый элемент списка.

ВХОД	ВЫХОД
5	8 8 2 2 1 1 4 4 2 2
8 2 1 4 2	

- На вход подаётся n чисел. Сохраните эти числа в связном списке. Удалите все чётные числа из списка и напечатайте его.

ВХОД	ВЫХОД
5	1 5
8 2 1 4 5	

Часть 6: Дополнительно об итераторах

Обратные итераторы

Проход по контейнеру в обратном порядке с использованием обычных итераторов может быть затруднителен. С обратными итераторами это сделать гораздо проще. Метод `rbegin` возвращает итератор на последний элемент. Метод `rend` возвращает итератор на фиктивный элемент, следующий до первого. Перегруженный оператор `++` перемещает итератор к предыдущему элементу.

```
#include <iostream>
#include <vector>
int main() {
    std::vector<int> v {4, 8, 15, 16, 23, 42};

    for (std::vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); it++) {
        std::cout << *it << " ";
    }
}
```

Константные итераторы

Рассмотрим следующую ситуацию:

```
#include <iostream>
#include <vector>
int main() {
    const std::vector<int> v{ 4, 8, 15, 16, 23, 42 };

    for (std::vector<int>::iterator it = v.begin(); it != v.end(); it++) {
        std::cout << *it << " ";
    }
}
```

Вектор `v` – константный, а его вызываемые методы `begin` и `end` константными не являются. Компилятор знает, что эти методы могут быть использованы для изменения вектора `v` и поэтому запрещает их использование. Решение – использование итератора `const_iterator` и методов `cbegin` и `cend`.

- Измените код выше чтобы он работал, поменяв обычный итератор на константный.
- Напишите функцию `printEven`, которая будет принимать вектор по константному указателю и печатать только чётные элементы.
- Напишите функцию `printOdd`, которая будет принимать множество по константному указателю и печатать только нечётные элементы. Можно ли в этом случае использовать обычные неконстантные итераторы.

Функция `std::copy`

Стандартная функция `std::copy` принимает на вход три итератора. Первые два итератора задают множество элементов которые нужно скопировать, а третий – место куда нужно их скопировать. Пример кода: (полная версия в файле `std_copy.cpp`).

```
std::vector<int> a { 4, 8, 15, 16, 23, 42 };
std::vector<int> b(a.size());
std::cout << b;

std::copy(a.begin(), a.end(), b.begin());
std::cout << b;
```

- Почему в конструктор вектора `b` передаётся `a.size()`? Что будет если его не передавать?
- Измените контейнер `a` с `vector` на `set` и скопируйте содержимое `a` в `b`.

Итератор `std::back_insert_iterator`

`std::back_insert_iterator<Container>` – это специальный итератор у которого операторы перегружены по другому. Для него:

- `operator*` ничего не делает
- `operator++` ничего не делает
- `operator=` вызывает метод `push_back` контейнера

Благодаря таким перегрузкам поведение этого итератора сильно отличается от поведения обычных итераторов. К примеру, следующий код добавит в контейнер `a` ещё один элемент:

```
std::vector<int> a { 1, 2, 3 };
std::back_insert_iterator<std::vector<int>> it{a};
*it = 4;
```

Так как тип этого итератора может иметь длинное название, то была введена функция под названием `std::back_inserter`, которая принимает на вход контейнер и возвращает такой итератор. Пример в котором вектор `a` копируется в пустой вектор `b` (полная версия в `std_copy_back_inserter.cpp`):

```
std::vector<int> a { 4, 8, 15, 16, 23, 42 };
std::vector<int> b;
std::cout << b;

std::copy(a.begin(), a.end(), std::back_inserter(b));
std::cout << b;
```

- Напишите функцию `append`, которая будет принимать 2 вектора. Первый вектор эта функция должна принимать по ссылке, а второй – по константной ссылке. Функция должна копировать всё содержимое второго вектора в первый с помощью функции `std::copy`.

Итератор `std::ostream_iterator`

`std::ostream_iterator<T>` – это специальный итератор у которого операторы перегружены по другому:

- `operator*` ничего не делает
- `operator++` ничего не делает
- `operator=` выводит соответствующий элемент в выходной поток (например, `std::cout` или файл) с помощью оператора `<<`
- Что сделает эта программа:

```
int main() {
    std::ostream_iterator<int> it{ std::cout, ", " };
    it = 1;
    *it = 2;
    it++ = 3;
}
```

- Напишите программу, которая печатает числа от 1 до 100, разделённые символом `+`. Используйте `ostream_iterator`.
- Напишите программу, которая печатает содержимое вектора на экран, используя `std::copy` и `ostream_iterator`.
- Пусть есть такое множество строк:

```
std::set<std::string> a{"Cat", "Dog", "Mouse", "Elephant"};
```

Напечатайте содержимое этого множества на экран, каждый элемент на новой строке.