# Задание по программированию: AirportCounter

## Условие

Вам необходимо разработать простейшую систему сбора статистики вылетов для небольших авиакомпаний. Высокоуровневые требования выглядят следующим образом:

1. Система сбора статистики должна представлять собой класс AirportCounter, умеющий в произвольный момент времени для заданного аэропорта вернуть текущее количество вылетов из него.
2. Набор возможных аэропортов фиксирован в виде enum и потому не может изменяться в рамках одного сеанса работы системы статистики, то есть в рамках одного экземпляра AirportCounter. Таким образом, класс AirportCounter необходимо шаблонизировать типом этого enum: если назвать такой enum TAirport, получим шаблонный класс AirportCounter<TAirport>.
3. Гарантируется (по крайней мере, обещано заказчиком), что по enum аэропортов можно будет легко определить количество элементов в нём, а также перебрать сами элементы. А именно, любой используемый enum будет иметь вид enum class SomeCityAirport { Airport1, Airport2, ..., AirportN, Last\_ };, то есть будет содержать некоторое количество элементов-аэропортов и элемент Last\_ в качестве последнего. Поскольку в этом enum не будут заданы числовые значения для элементов, можно полагаться на то, что они будут пронумерованы подряд от 0 до N-1, а выражение static\_cast<uint32\_t>(SomeCityAirport::Last\_) будет содержать количество аэропортов (N).
4. Класс должен уметь конструироваться по списку аэропортов вылета, а также позволять обновлять этот список: добавить вылет из какого-то аэропорта, удалить один вылет (например, ошибочно добавленный) или удалить из истории все вылеты из некоторого аэропорта.

## Тестирование

Прежде чем перейти к более детальному описанию интерфейса класса, опишем процесс тестирования задачи.

Вам **будут даны все тесты,** на которых будет проверяться шаблонный класс AirportCounter. Ваша задача — добиться не только корректной работы на этих тестах, но и уложиться на них в **2 секунды.** (Считайте, что заказчик предоставил вам подробнейшие сценарии использования вашего класса, которые затем были преобразованы в конкретные бенчмарки.)

Таким образом, вы должны выбрать способ реализации класса, ориентируясь лишь на конкретные бенчмарки.

Отметим, что данные бенчмарки составлены на основе реального примера из высоконагруженного сервиса, замедление которого даже на десятки процентов недопустимо. Именно поэтому в этой задаче важно не только спроектировать решение с приемлемой асимптотической сложностью, но и «разогнать» его на конкретных примерах.

## Интерфейс класса

template <typename TAirport>

class AirportCounter {

public:

  // конструктор по умолчанию: список элементов пока пуст

  AirportCounter();

  // конструктор от диапазона элементов типа TAirport

  template <typename TIterator>

  AirportCounter(TIterator begin, TIterator end);

  // получить количество элементов, равных данному

  size\_t Get(TAirport airport) const;

  // добавить данный элемент

  void Insert(TAirport airport);

  // удалить одно вхождение данного элемента

  void EraseOne(TAirport airport);

  // удалить все вхождения данного элемента

  void EraseAll(TAirport airport);

  using Item = pair<TAirport, size\_t>;

  // получить некоторый объект, по которому можно проитерироваться,

  // получив набор объектов типа Item - пар (аэропорт, количество),

  // упорядоченных по аэропорту

  ??? GetItems() const;

private:

  // ???

};

## Файл с заготовкой решения и тестами

[airport\_counter.cpp](https://d3c33hcgiwev3.cloudfront.net/5zBNao5FEeiEShL6YrwsWg_e7740c708e4511e8bccc6fe6f3f3ac49_airport_counter.cpp?Expires=1617062400&Signature=Z4IKdBqjOOwZMHmq8jx8IBk~v8zj5V~MUDGB8UuasaHnlN8pygtHoZ-oBbs2PeTMhX-NihJ9bqFVcRsu7dQLYL-7fw17hUK4oAKmM4IP~1r-d8j8zD-O5JMvzuvmcITBd4FlTBybuMb2oYp2PCKYp38k3~qASToVPa-ISH47XVs_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

## Подсказки

Если задача вызывает непреодолимые трудности, вы можете воспользоваться подсказками. Они помогут быстрее сдать задачу, но в итоге вы получите меньше пользы от процесса поиска решения.

Не нужно открывать все подсказки сразу. Используйте их последовательно.

## Подсказка 1

Внимательно изучите бенчмарки. Рассмотрите все изученные контейнеры на предмет того, можно ли их использовать в данной задаче. Выпишите плюсы и минусы каждого из них. В конечном счёте, внимания заслуживает лишь один контейнер

## Подсказка 1

Используйте массив. Рассмотрите разные варианты того, что именно должно храниться в массиве. Не поддавайтесь соблазну реализовать лишь одну идею (кажущуюся наиболее оптимальной) и улучшать её в случае непрохождения тестов. Попробуйте разные подходы, даже — на первый взгляд — неоптимальные. Один из них (довольно простой) и приведёт вас к успеху. Это в целом соответствует реальному опыту: все идеи нужно проверять на данных, специфичных для данного сервиса. Оптимальность на случайных запросах ничего не стоит, если сервис тормозит в продакшене.