

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: `std::set`

`std::set` — это реализация множества с помощью бинарного дерева поиска. Не хранит дубликатов. При попытке добавить в множество тот элемент, который в нём уже есть, ничего не произойдёт. Также все элементы в множестве всегда хранятся в отсортированном виде (так как это бинарное дерево поиска). Для типа элементов множество должен быть реализован `operator<`. В `std::set` нельзя менять элементы, так как это бинарное дерево поиска, но можно удалить элемент, а потом вставить новый.

Основные методы для работы с множеством:

метод	описание
<code>insert</code>	Вставляет элемент в множество
<code>erase</code>	Удаляет элемент. Можно удалять по значению элемента или по итератору. Также можно сразу удалить диапазон значений, если передать 2 указателя
<code>find(x)</code>	Принимает на вход значение <code>x</code> и ищет такой элемент в множестве. Возвращает итератор на этот элемент или итератор <code>end()</code> , если такого элемента нет
<code>count(x)</code>	Принимает значение и находит, сколько элементов равны этому значению (т.е. 0 или 1)
<code>lower_bound(x)</code> <code>upper_bound(x)</code>	Возвращает итератор на первый элемент, который больше или равен <code>x</code> Возвращает итератор на первый элемент, который больше <code>x</code>

- На вход подаётся n чисел. Напечатайте эти числа удалив все дубликаты.

ВХОД	ВЫХОД
10	1 2 7 8
8 2 1 2 2 1 8 7 1 2	

- На вход подаётся n чисел и некоторое число `x`. Найдите пару элементов массива, такую что их сумма равна `x`. Напечатайте индексы этих элементов. При наличии нескольких таких пар, напечатайте любую. Решение должно работать за $O(n \log(n))$ или быстрее.

ВХОД	ВЫХОД
8	2 4
8 2 5 4 9 1 7 4	
14	

`std::multiset`

То же самое, что и `std::set`, но может хранить дубликаты. Одна из неочевидных особенностей `multiset` это то, что при удалении элемента по значению `erase(x)`, удалятся все элементы, равные `x`. Для удаления одного элемента нужно передать в `erase` итератор на элемент.

- Считайте n чисел и отсортируйте их с помощью вставки в `multiset`. Распечатайте отсортированные числа.
- На прямой лежит верёвка длиной n метров. Затем её начинают последовательно разрезать. Все места разрезов — целые числа. Найти длину самого длинного куска после каждого разреза.

ВХОД	ВЫХОД
20 8	12 10 8 7 6 5 5 4
8 10 15 1 7 4 11 18	

Часть 3: std::map

`std::map` – это реализация словаря с помощью бинарного дерева поиска. Не хранит ключе - дубликатов. При попытке добавить в этот словарь элемента с ключом, который в нём уже есть, ничего не произойдёт. Также все элементы в этом словаре всегда хранятся в отсортированном по ключам виде (так как это бинарное дерево поиска). Для типа ключей должен быть реализован `operator<`. В `std::map` можно менять значения, но нельзя менять ключи, так как это бинарное дерево поиска. Но можно удалить элемент каким-то ключом, а потом вставить новый с другим ключом.

Основные методы для работы с множеством:

метод	описание
<code>insert(k, v)</code>	Вставляет элемент с ключом <code>k</code> и значением <code>v</code> Если такой элемент уже есть, то ничего не делает
<code>operator[]</code> <code>m[k] = v</code>	Вставляет элемент с ключом <code>k</code> и значением <code>v</code> Если такой элемент уже есть, то меняет его значение
<code>erase(k)</code>	Удаляет элемент. Можно удалять по значению элемента или по итератору. Также можно сразу удалить диапазон значений, если передать 2 указателя
<code>find(x)</code>	Принимает на вход значение <code>x</code> и ищет элемент с таким ключом. Возвращает итератор на этот элемент или итератор <code>end()</code> , если такого ключа нет
<code>count(x)</code>	Принимает значение и находит, сколько ключей равны этому значению (т.е. 0 или 1)
<code>lower_bound(x)</code> <code>upper_bound(x)</code>	Возвращает итератор на первый элемент, который больше или равен <code>x</code> Возвращает итератор на первый элемент, который больше <code>x</code>

Пример программы, которая создаёт словарь из пар <название города, его население>. Строка выступает в качестве ключа, а целое число – в качестве значения.

```
#include <iostream>
#include <string>
#include <map>
using std::cout, std::endl;

int main ()
{
    std::map<string, int> m = {{"London", 8900000}, {"Moscow", 12500000}, {"Milan", 4300000}};

    std::string cityName;
    while (true)
    {
        std::cin >> cityName;
        if (cityName == "q" || cityName == "quit")
            break;

        std::map<std::string, int>::iterator it = m.find(cityName);
        if (it == m.end())
            cout << "No such city" << endl;
        else
            cout << "City " << cityName << " population = " << it->second << endl;
    }
}
```

- Напишите программу, которая будет в бесконечном цикле считывать слова и после каждого считывания печатать все уникальные слова, считанные ранее и количество таких слов. Например, если пользователь ввёл слово **Cat** три раза, слово **Dog** 1 раз и слово **Elephant** 2 раза. То после очередного считывания программа должна напечатать:

Dictionary:

Cat: 3

Dog: 1

Elephant: 2

- Считайте все слова из файла и напечатайте все уникальные слова и то, как часто они встречались в файле. Сохраните результат в новом файле. Для работы с файлами можно использовать функции C.

входной файл	выходной файл
I'm having Spam, Spam, Spam, Spam, Spam, Spam, Spam, baked beans, Spam, Spam, Spam and Spam.	I'm 1 Spam 1 Spam, 9 Spam. 1 and 1 beans, 1 having 1