Выйти

# Задача на ревью для третьего длинного контеста

| | |
|---|---|
| | 21 янв 2023, 15:17:40 |
| старт: | 10 янв 2023, 04:00:00 |
| финиш: | 30 янв 2023, 02:00:00 |
| до финиша: | 8д. 10ч. |
| начало: | 10 янв 2023, 04:00:00 |
| конец: | 30 янв 2023, 02:00:00 |
| длительность: | 19д. 22ч. |

## A. Хеш-таблица

| | |
|---|---|
| Ограничение времени | 2 секунды |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Вам предстоит реализовать аналог контейнера std::unordered_map — ассоциативный массив на основе хеш-таблицы, который поддерживает отображение ключ -> значение, т.е. в соответствие каждому ключу ставится единственное значение, а все ключи уникальны.
Вы должны написать шаблон HashMap, который параметризуется типом ключа, типом значения и типом «хешера», иными словами, следующее:

template<class KeyType, class ValueType, class Hash = std::hash<KeyType> > class HashMap;

Скажем пару слов про «хешер». Это некоторый тип, который по ключу типа KeyType умеет выдавать значение типа size_t, которое можно получить, используя функциональный вызов. Например:

```
// пусть hasher имеет тип Hash
KeyType key = ...; // какой-то ключ
size_t num = hasher(key);
```

Благодаря использованию шаблонов, тип Hash может быть чем угодно — функцией, лямбдой или же классом, для которого перегружен оператор вызова (). Это позволяет пользователю вашего класса выбирать наиболее предпочтительный для него вариант хеш-функции. Если же пользователя устраивает стандартный вариант, то используется стандартный тип std::hash<KeyType>.

Обратите внимание, что сам по себе хешер всего лишь дает вам возможность получить для любого объекта некоторое число (и предоставляется пользователем класса, поскольку вы заранее не знаете, с какими типами будет использоваться ваша таблица), а вот как его использовать для организации быстрой хеш-таблицы — уже ваша забота. Однако же если хешер, например, возвращает для всех ключей число 0, то ясно, что это проблема пользователя, а от вас мало что зависит (тем не менее, ваш класс должен по-прежнему корректно работать в таких ситуациях, это должно отражаться только на времени работы). Поэтому вы можете считать, что хешер распределяет ключи по диапазону size_t достаточно равномерно (в предположении, что ключи случайны) — в частности, это верно для умолчательного варианта std::hash<KeyType>.

Ваш класс должен содержать следующие конструкторы и методы:

1. Конструктор по умолчанию.
2. Конструктор, принимающий итераторы на начало и конец (точнее, следующий за концом) последовательности, каждый элемент которой представляет собой std::pair<ключ, значение>.
3. Конструктор, принимающий std::initializer_list описанных выше пар.
4. Все конструкторы также должны поддерживать возможность передачи объекта хешера (т.е. имеющего тип Hash) последним аргументом. В случае же, если таковой не передается, то используйте сконструированный по умолчанию.
5. Методы size и empty, которые должны быть константными, и которые возвращают количество элементов в таблице и пуста ли она соответственно.
6. Константный метод hash_function, который возвращает используемый таблицей хешер по значению.
7. Метод insert, который принимает std::pair<ключ, значение> и добавляет в таблицу связь ключ -> значение. Если данный ключ уже имеется в таблице, то метод не должен ничего делать.
8. Метод erase, который принимает ключ и удаляет соответствующую пару (ключ, значение) из таблицы. Если искомого ключа нет, то метод не должен ничего делать.
9. Ваш класс должен предоставлять типы iterator и const_iterator, соответствующие итератору и константному итератору, с помощью которых можно было бы просмотреть содержимое таблицы, а также соответствующие методы begin и end для обоих типов итераторов, возвращающих итератор на начало контейнера и следующий за последним (как все контейнеры стандартной библиотеки). Итератор должен адресовывать значения типа std::pair<const KeyType, ValueType>&, а константный итератор — типа

const std::pair<const KeyType, ValueType>&. Таким образом, с помощью обычного итератора можно изменять только значения в таблице (но не ключи), а с помощью константного итератора модификации невозможны совсем.

Обратите внимание, что итерирование по всей таблице должно занимать линейное время по числу вставленных элементов. Однако порядок, в котором перебираются элементы таблицы, может быть произвольным.

Итераторы должны быть, по крайней мере, forward — грубо говоря, поддерживать конструирование, инкремент, разыменовывание (через * и ->), а также операторы сравнения == и !=. Более подробно можно ознакомиться тут: http://www.cplusplus.com/reference/iterator/ForwardIterator/.

Ваш класс может инвалидировать все итераторы после вставок и удалений. Иными словами, если из таблицы удаляется или вставляется элемент, то все имеющиеся на данный момент итераторы могут стать недействительными. Стандартный std::unordered_map при вызове erase инвалидирует только итераторы на удаленный элемент, остальные итераторы остаются действительными.

10. Метод find, константный (возвращающий const_iterator) и нет (возвращающий iterator), который по переданному ключу возвращает итератор на соответствующую пару (ключ, значение), либо end(), если искомого ключа нет в таблице.

11. Оператор [ ], который по переданному ключу возвращает ссылку на соответствующее значение. Если же искомого ключа в таблице нет, то метод должен добавить в таблицу пару (ключ, значение по умолчанию) и вернуть соответствующую ссылку. Таким образом, можно будет писать что-то вроде

HashMap<int, int> table;
table[3] = 5;
std::cout « table[3]; // выведет 5

12. Константный метод at, который работает аналогично оператору [ ], но возвращает константную ссылку на значение, а при отсутствии ключа генерирует исключение типа std::out_of_range.

13. Метод clear, который очищает таблицу, удаляя все вставленные элементы. Обратите внимание, что метод должен работать за линейное время по количеству элементов в таблице.

Для сравнения ключей используйте только оператор ==.

Если вы используете ручное управление памятью (например, вручную выделяете память через new вместо использования std::vector), то, разумеется, ваш класс также должен предоставлять правильно определенные конструктор копирования, оператор присваивания и деструктор.

## Формат ввода

Всего будет не более $10^6$ операций с таблицей.

## Пример

| Ввод | Вывод |
|---|---|
| 13<br>+ 3 5<br>+ 2 1<br>+ 0 7<br>? 0<br>- 0<br>? 0<br>? 2<br>+ 8 -4<br><<br>!<br>? 3<br>+ 3 3<br>? 3 | 7<br>-1<br>1<br>3 5<br>2 1<br>8 -4<br>-1<br>3<br>1 |

## Примечания

Вы должны прислать заголовочный файл, содержащий определение вашего класса. Обратите внимание, что тестирующая программа достаточно строго проверяет требования из условия. В случае их несоблюдения вы будете получать вердикт Compile Error (или WA/PE/RE на 1 тесте).
Помимо стандартных тестов контеста запускаются следующие юнит-тесты: https://gist.github.com/astiunov/bd453be581723a95f8502844168c7e16.

В примере выше во входных данных вводится число запросов к таблице, далее следуют сами запросы:

+ key value обозначает map[key] = value
- key — вызов map.erase(key)
? key — вывод -1, если ключа нет, и соответствующего данному ключу значения в противном случае.
< — вывод содержимого таблицы в формате «ключ значение».
! — вызов метода clear.

Последнее число в выходных данных — количество элементов в таблице после всех операций.

Язык (make) GCC C++17

Набрать здесь | Отправить файл

1

Отправить