Введение

В курсе «Белый пояс по С++» [финальным заданием](https://www.coursera.org/learn/c-plus-plus-white/programming/sO7Vq/final-naia-zadacha-kursa) была разработка базы данных, которая работала с парами (*дата*, *событие*). В этой задаче вам нужно развить эту базу данных, научив её выбирать и удалять события, удовлетворяющие заданному условию, а также разделив её код на несколько файлов.

Более подробно, ваша программа должна уметь обрабатывать набор команд:

* **Add *date* *event*** — добавить в базу данных пару (***date***, ***event***);
* **Print** — вывести всё содержимое базы данных;
* **Find *condition*** — вывести все записи, содержащиеся в базе данных, которые удовлетворяют условию ***condition***;
* **Del *condition*** — удалить из базы все записи, которые удовлетворяют условию ***condition***;
* **Last *date*** — вывести запись с последним событием, случившимся не позже данной даты.

Условия в командах **Find** и **Del** накладывают определённые ограничения на даты и события, например:

* **Find date < 2017-11-06** — найти все события, которые случились раньше 6 ноября 2017 года;
* **Del event != "holiday"** — удалить из базы все события, кроме «**holiday**»;
* **Find date >= 2017-01-01 AND date < 2017-07-01 AND event == "sport event"**— найти всё события «**sport event**», случившиеся в первой половине 2017 года;
* **Del date < 2017-01-01 AND (event == "holiday" OR event == "sport event")** — удалить из базы все события «**holiday**» и «**sport event**», случившиеся до 2017 года.

В командах обоих типов условия могут быть пустыми: под такое условие попадают все события.

Структура программы

Ниже вам даны заготовки для файлов

* condition\_parser.h/cpp — в видеолекции «[Задача разбора арифметического выражения. Описание решения](https://www.coursera.org/learn/c-plus-plus-yellow/lecture/rVRa6/zadacha-razbora-arifmietichieskogho-vyrazhieniia-opisaniie-rieshieniia)» мы продемонстрировали построение абстрактного синтаксического дерева для арифметических выражений. Реализация этого алгоритма для разбора условий в командах **Find** и **Del**содержится в функции ParseCondition, объявленной и полностью реализованной в файлах condition\_parser.h/cpp;
* token.h/cpp — содержат готовый токенизатор, который используется в функции ParseCondition;
* main.cpp — содержит готовую функцию main.

Вам нужно проанализировать выданный код и разработать недостающие классы и функции:

* класс Database, который представляет собой базу данных, — вы должны сами создать его публичный интерфейс, основываясь на том, как он используется в функции main;
* классы Node, EmptyNode, DateComparisonNode, EventComparisonNode и LogicalOperationNode — сформировать их иерархию и публичный интерфейс вам поможет анализ функций main и ParseCondition;
* класс Date, а также функцию ParseDate и оператор вывода в поток для класса Date.

На проверку вы должны прислать архив, состоящий из файлов:

* date.h/cpp — эти файлы должны содержать объявления и определения класса Date, функции ParseDate и оператора вывода в поток для класса Date;
* database.h/cpp — эти файлы должны содержать объявление и определение класса Database;
* node.h/cpp — эти файлы должны содержать объявления и определения класса Node, а также всех его потомков (см. выше), которые представляют собой узлы абстрактного синтаксического дерева, формируемого функцией ParseCondition;
* condition\_parser.h/cpp;
* token.h/cpp;
* main.cpp;
* другие .h- и .cpp-файлы, которые вы посчитаете нужным создать в своём решении.

Как будет тестироваться ваше решение

Тестирование вашего решения будет выполняться в два этапа. На первом этапе автоматическая тестирующая система распакует присланный вами архив и соберёт извлечённые файлы в исполняемый файл. Затем этот исполняемый файл будет запущен на наборе тестов. Тестирование выполняется так же, как и для большинства задач на нашем курсе — тест подаётся в stdin, замеряется время выполнения программы, а затем анализируется stdout.

На втором этапе будет выполняться тестирование отдельных файлов вашего проекта. Проверяется, что файл date.h действительно содержит объявление класса Date, что Database::FindIf корректно выполняет поиск по переданному предикату и т.д. Мы ожидаем от ваших классов интерфейс, который зафиксирован в функции main. Поэтому в классах Database и Node делайте у методов именно те сигнатуры, которые используются в функции main.

Формат ввода и вывода

В стандартном вводе содержатся команды для работы с базой данных, по одной команде в строке. Ваша программа должна считать их и вывести результаты обработки в стандартный вывод. Правила обработки команд приведены ниже.

Команда Add

Встретив команду **Add *date* *event***, ваша программа должна добавить пару (***date***, ***event***) в базу данных и затем показывать её при поиске (команда **Find**) или печати (команда **Print**). Одинаковые события, произошедшие в один и тот же день, сохранять не нужно: добавление в базу уже существующей пары (***date***, ***event***) должно игнорироваться. В одну и ту же дату может произойти много разных событий, БД должна суметь их все сохранить.

Гарантируется, что поле ***date*** в команде **Add** имеет формат «*Год*-*Месяц*-*День*», где *Год* — это целое число от 0 до 9999, *Месяц* — это номер месяца от 1 до 12 включительно, *День* — это номер дня от 1 до 31 включительно. После даты обязательно следует пробел, отделяющий её от события. Примеры корректных дат: **2017-11-07**, **0-2-31**.

Вся остальная часть строки в команде **Add** задаёт событие. Оно может содержать пробелы, поэтому для его считывания удобно воспользоваться функцией getline. При этом гарантируется, что название события не может содержать символ кавычки (**"**). Таким образом, этот символ может встретиться лишь в условии в командах **Find** и **Del**, являясь ограничителем названия события.

Пример

Для команды **Add 2017-11-07 big sport event** программа должна добавить в базу данных пару (**2017-11-07**, **big sport event**).

Команда Del

Встретив команду **Del *condition***, ваша программа должна удалить из базы данных все события, удовлетворяющие условию ***condition***, и вывести в стандартный вывод количество удалённых записей ***N*** в формате «**Removed *N*entries**». Если условие пусто, результатом выполнения команды должна стать очистка всей базы данных.

Пример

Ввод



1

2

3

4

5

Add 2017-06-01 1st of June

Add 2017-07-08 8th of July

Add 2017-07-08 Someone's birthday

Del date == 2017-07-08

Вывод



1

2

Removed 2 entries

Команда Print

Встретив команду **Print**, ваша программа должна вывести все пары (*дата*, *событие*), которые в данный момент содержатся в базе данных. Пары надо выводить по одной в строке. Они должны быть отсортированы по дате по возрастанию. События в рамках одной даты необходимо выводить в порядке добавления (за исключением уже удалённых и попыток добавления повторов).

Даты надо выводить в формате **ГГГГ-ММ-ДД**, где Г, М, Д — это цифры чисел года, месяца и дня соответственно. Если какое-то число имеет меньше разрядов, чем нужно, его надо дополнить нулями, например: **0001-01-01** — первое января первого года.

Пример

Ввод



1

2

3

4

5

6

Add 2017-01-01 Holiday

Add 2017-03-08 Holiday

Add 2017-1-1 New Year

Add 2017-1-1 New Year

Print

Вывод



1

2

3

4

2017-01-01 Holiday

2017-01-01 New Year

2017-03-08 Holiday

*Обратите внимание, что событие "New Year" выведено только один раз, несмотря на то, что оно добавлено дважды. Как сказано в описании команды Add, одинаковые события, произошедшие в один и тот же день, сохранять не нужно.*

Команда Find

Встретив команду **Find *condition***, ваша программа должна вывести все пары (*дата*, *событие*), которые в данный момент содержатся в базе данных и удовлетворяют условию ***condition***. Формат вывода аналогичен команде **Print**. Поиск с пустым условием эквивалентен команде **Print**. В конце команда должна вывести строку **Found *N* entries**, где ***N***— количество найденных пар.

Пример

Ввод



1

2

3

4

5

6

Add 2017-01-01 Holiday

Add 2017-03-08 Holiday

Add 2017-01-01 New Year

Find event != "working day"

Add 2017-05-09 Holiday

Вывод



1

2

3

4

5

2017-01-01 Holiday

2017-01-01 New Year

2017-03-08 Holiday

Found 3 entries

Команда Last

По команде **Last *date*** нужно вывести последнее из событий, случившихся к дате ***date***. Более формально:

* среди всех имеющихся дат событий нужно найти наибольшую, не превосходящую ***date***;
* из всех событий с такой датой нужно выбрать **последнее добавленное** и вывести в формате, аналогичном формату команды **Print**;
* если ***date*** меньше всех имеющихся дат, необходимо вывести «**No entries**».

Пример

Ввод



1

2

3

4

5

6

7

8

Add 2017-01-01 New Year

Add 2017-03-08 Holiday

Add 2017-01-01 Holiday

Last 2016-12-31

Last 2017-01-01

Last 2017-06-01

Add 2017-05-09 Holiday

Вывод



1

2

3

4

No entries

2017-01-01 Holiday

2017-03-08 Holiday

Пример

Ввод



1

2

3

4

5

6

7

8

9

10

Add 2017-11-21 Tuesday

Add 2017-11-20 Monday

Add 2017-11-21 Weekly meeting

Print

Find event != "Weekly meeting"

Last 2017-11-30

Del date > 2017-11-20

Last 2017-11-30

Last 2017-11-01

Вывод



1

2

3

4

5

6

7

8

9

10

11

2017-11-20 Monday

2017-11-21 Tuesday

2017-11-21 Weekly meeting

2017-11-20 Monday

2017-11-21 Tuesday

Found 2 entries

2017-11-21 Weekly meeting

Removed 2 entries

2017-11-20 Monday

No entries

Требования к эффективности операций

* Операции **Print**, **Find** и **Del** в худшем случае затрагивают всю базу, поэтому не обязаны быть максимально эффективными. Считайте, что каждая из этих операций имеет право тратить одно или несколько действий на каждое существующее в базе событие.
* Операции **Add** и **Last** должны быть эффективными: каждая из них может использовать несколько лишь несколько простых операций вроде добавления в set, поиска ключа в map или удаления последнего элемента из вектора. Если для обработки операции **Add** и **Last** используется цикл, вероятнее всего, код недостаточно эффективен и не будет принят проверяющей системой.