ip_filter

Условие

Программа из стандартного ввода читает данные. Данные хранятся построчно. Каждая строка состоит из трех полей разделенных одним символом табуляции и завершается символом конца строки.

```
text1 \t text2 \t text3 \n
```

Поля *text2* и *text3* игнорируются. Поле *text1* имеет следующую структуру (ip4 address):

```
n1.n2.n3.n4
```

где n1..4 - число от 0 до 255

Необходимо загрузить список ip-адресов в память и вывести в следующем порядке:

Список адресов отсортированных в обратном лексикографическом порядке адресов. Одна строка один адрес.

Пример лексикографической сортировки (по первому числу, затем по второму и так далее):

```
1.1.1.1
1.2.1.1
1.10.1.1
```

Соответственно обратная:

```
1.10.1.1
1.2.1.1
1.1.1.1
```

Сразу продолжается список адресов первый байт которых равен 1. Порядок сортировки не меняется. Одна строка один адрес. Списки ничем не разделяются.

Сразу продолжается список адресов первый байт которых равен 46, а второй 70. Порядок сортировки не меняется. Одна строка один адрес. Списки ничем не разделяются.

Сразу продолжается список адресов любой байт которых равен 46. Порядок сортировки не меняется. Одна строка один адрес. Списки ничем не разделяются.

Требования к реализации

В приложенном к задании исходном файле необходимо заменить, где это возможно конструкции на аналогичные из стандарта C++14.

Реализовать недостающий функционал.

Выдержка из файла со списком адресов к задаче прилагается.

Лишний раз проверьте

- лексикографическая сортировка понятна как для строки так и для контейнера
- выбрана соответствующая задаче структура данных

Результат

- пакет `ip_filter` содержащий исполняемый файл `ip_filter` опубликован на Bintray
- отправлена на проверку ссылка на страницу решения (github)

Проверка

Задание считается выполнено успешно, если после просмотра кода, подключения репозитория, установки пакета и запуска бинарного файла командой:

```
cat ip_filter.tsv | ip_filter
```

Вывод совпадет с `ip_filter.tst`.

Оценка

баллы	за что
5	Успешно прошла проверка.
2	Логарифмическая сложность одного из фильтров.
1	Сортировка без написания своих лямбд и операторов сравнения.
2	Не написали своих аналогов для `std::any_of`, `std::equal`, `std::copy_if`.
1	Используются синонимы типов через `using`.
2	Один из фильтров реализован как variadic template.
2	Фильтры реализованы как лямбды.
2	Написаны и работают тесты.
1	Нет ни одного предупреждения от компилятора.