

Задание 1

Задание состоит из двух частей – создание сборочной фермы и реализация приложения для фильтрации IP-адресов. Части почти что независимые – только для сдачи второй потребуется выполнить первую.

Рекомендуемый порядок выполнения – первым этапом разобраться с работой сборочной фермы на простейшем примере (приложение “Hello, World!”), затем использовать настроенную сборочную ферму для построения приложения фильтрации IP-адресов.

Варианты организации исходником по самостоятельным работам:

- отдельный репозиторий под каждую работу
- отдельная ветка в одном и том же репозитории под каждую работу
- отдельная директория в одной и той же ветке одного и того же репозитория

[продолжение на следующем листе]

Часть 1. Создание сборочной фермы

Написать программу, выводящую на консоль две строки:

```
build N  
Hello, World!
```

Где вместо N должен выводиться текущий номер сборки. Запустить на этапе сборки тесты, проверяющие валидность номера версии. Выложить исходные тексты в репозиторий на github.

Настроить [Github Actions workflow](#) для автоматической сборки проекта на каждый коммит. Так же настроить автоматический выпуск релиза либо на каждый коммит, либо по созданию тэга.

Самоконтроль

- версия пакета увеличивается от сборки к сборке
- актуальная версия выводится в приветственном сообщении
- пакет `helloworld`, содержащий исполняемый файл `helloworld`, опубликован в качестве релиза в репозитории

Проверка

Задание считается выполненным успешно, если после установки скаченного из релиза пакета:

```
apt update && apt install -y helloworld-0.0.X-Linux.deb
```

(вместо X – номер билда), запуска бинарного файла:

```
helloworld_cli
```

появилось сообщение:

```
Version: X  
Hello, World!
```

(вместо X – опять же, номер билда).

Дополнительное упражнение

Добавить unit-тест, который будет полностью повторять предложенный в материалах вебинара тест `test_version.cpp`, но сделать его с использованием [GoogleTest framework](#). GoogleTest при этом либо подключить к репозиторию в качестве submodule и собрать вместе с тестом, либо установить на этапе подготовке к сборке через Github Actions.

[продолжение на следующем листе]

Часть 2. Фильтрация IP-адресов

Программа из стандартного ввода читает данные. Данные хранятся построчно. Каждая строка состоит из трех полей, разделенных одним символом табуляции, и завершается символом конца строки. Формат 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 – целое число от 0 до 255.

Требуется загрузить список ip-адресов в память и отсортировать их в обратном лексикографическом порядке. 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
```

Обратите внимание – сортировка выполняется в байтовом (численном) представлении IP-адреса, а не в строковом. Поэтому адрес, например, 1 . 2 . 1 . 1 меньше адреса 1 . 10 . 1 . 1 (ведь $2 < 10$).

Далее выводим в стандартный вывод следующее:

1. Полный список адресов после сортировки. Одна строка - один адрес.
2. Сразу следом список адресов, первый байт которых равен 1. Порядок сортировки не меняется. Одна строка - один адрес. Списки ничем не разделяются.
3. Сразу продолжается список адресов, первый байт которых равен 46, а второй 70. Порядок сортировки не меняется. Одна строка - один адрес. Списки ничем не разделяются.
4. 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 заданию исходном файле необходимо заменить, где это возможно, конструкции на аналогичные из стандарта C++14/C++17. Реализовать недостающий функционал.

Не обязательно следовать приложенному шаблону. Можно выполнить задание, оформив код любым корректным способом.

Лишний раз проверьте

1. лексикографическая сортировка понятна как для строки, так и для контейнера
2. выбрана соответствующая задаче структура данных

[продолжение на следующем листе]

Самопроверка

Макет исходного кода, а также тестовый файл с данными ip_filter.tsv прилагается к материалам занятия. Проверить себя можно следующим образом (Linux):

```
cat ip_filter.tsv | ip_filter | md5sum
```

```
24e7a7b2270daee89c64d3ca5fb3da1a -
```

Внимание! Из-за различного толкования символа '\n' на Linux и Windows, вышеприведенный хэш актуален только для Linux-систем. В случае, если Ваша рабочая система Windows, придумайте способ, как проверить реализацию, имея эталонный результат только для Linux.

Пакет `ip_filter`, содержащий исполняемый файл `ip_filter`, должен быть опубликован в качестве релиза в репозитории.

Дополнительное упражнение

Добавить unit-тесты для реализации на понравившемся тестовом framework-е (рекомендуем GoogleTest).

[вот теперь конец ☺]