# Учёт зверей

*Давайте напишем решение типовой задачи "мимо нас проходит поток чего-то разнородного, который нужно классифицировать и пересчитывать".*

*Хинт - задача имеет решение буквально в 10 строк, в лекциях был очень похожий пример. Для осознанного решения придётся немного почитать документацию соответственного STL-ного контейнера. Это нормально, это правильный и полезный опыт. Решить без осознания происходящего тоже можно, но лучше всё-таки разобраться.*

Похожая задача уже была, но в этот раз требования новые. Снова есть зоопарк разных классов с вот таким общим интерфейсом:

```
class Animal {
public:
    virtual string getType() const = 0;
    virtual ~Animal() {}
};
```

Смотритель зоопарка теперь должен стать более внимательным бюрократом и начать вести аккуратный учёт зверей - сколько и каких у него есть. Плюс уметь быстро ответить, сколько и кого у него в зоопарке. Прототип класса смотрителя зоопарка теперь вот такой:

```
class ZooKeeper {
public:
    // Создаём смотрителя зоопарка
    ZooKeeper();

    // Смотрителя попросили обработать очередного зверя.
    void handleAnimal(const Animal& a);

    // Возвращает, сколько зверей такого типа было обработано.
    // Если таких не было, возвращает 0.
    int getAnimalCount(const string& type) const;
};
```

Пример ожидаемого сценария работы:

```
    ZooKeeper z;
    Animal *a = new Monkey();
    z.handleAnimal(*a);
    delete a;
    a = new Monkey();
    z.handleAnimal(*a);
    delete a;
    a = new Lion();
    z.handleAnimal(*a);
    delete a;
    cout << z.getAnimalCount("monkey") << endl;
    cout << z.getAnimalCount("lion") << endl;
    cout << z.getAnimalCount("cat") << endl;
```

Должно напечатать:

```
2
1
0
```

Потому что обезьян было две, лев один, а котиков не было совсем.

# Пользовательские сессии

*Попробуйте на задаче, приближенной к реальности, ещё раз лично оценить - STL это удобно, или лучше всё писать самостоятельно с нуля.*

## Легенда

У нас есть нагруженная многопользовательская система. Хочется следить за тем, сколько пользователей активно в данный момент. *(Кто именно активен - в рамках данной задачи следить не требуется, но в реальной жизни это требование возникнет примерно сразу после того, как будет выполнен подсчёт количества.)*

## Постановка задачи

Напишите класс, мониторящий вход и выход пользователей, вот с таким прототипом:

```
class SessionManager
{
public:
    // Вход пользователя.
    // Один пользователь может войти несколько раз подряд,
    // считать его при этом нужно один раз.
    void login(const string& username);

    // Выход пользователя.
    // Пользователь может выйти несколько раз подряд,
    // падать при этом не нужно.
    void logout(const string& username);

    // Сколько сейчас пользователей залогинено.
    int getNumberOfActiveUsers() const;
};
```

При каждом входе пользователя наш класс уведомляют об этом, вызвав метод login. В этот момент нужно учесть, что данный пользователь стал активен. При выходе пользователя вызывается метод logout, нужно так или иначе вычеркнуть пользователя из активных.

Пример ожидаемого сценария работы:

```
    SessionManager m;
    m.login("alice");
    cout << m.getNumberOfActiveUsers() << endl;
    m.login("bob");
    cout << m.getNumberOfActiveUsers() << endl;
    m.login("alice");
    cout << m.getNumberOfActiveUsers() << endl;
    m.logout("whoever");
    cout << m.getNumberOfActiveUsers() << endl;
    m.logout("alice");
    cout << m.getNumberOfActiveUsers() << endl;
    m.logout("bob");
    cout << m.getNumberOfActiveUsers() << endl;
```

Будьте внимательны к ситуациям вида "повторный логин того же пользователя" или "попытка выхода того, кто никогда не входил"! В данном примере должна получиться вот такая статистика:

```
1
2
2
2
1
0
```

# Активность пользователей

*Ещё одна задача для оценки на личном опыте, стоит пользоваться STL, или лучше всё писать самостоятельно с нуля.*

## Легенда

В прошлой серии мы успешно решили задачу мониторинга пользовательских сессий (учёт входа и выхода пользователя). Этот успех так всех вдохновил, что теперь поставлена новая задача - собирать статистику активности каждого пользователя в системе. *(Задача, разумеется, упрощена относительно того, что захочется в реальности. Из самого заметного - в данной задаче в учёте активности нет разбиения на сессии, что в реальной жизни было бы очень странно.)*

## Постановка задачи

Напишите класс, мониторящий клики пользователей, вот с таким прототипом:

```
class Tracker
{
public:
    // При любом действии пользователя вызывается этот метод.
    // Методу передаётся:
    // - какой пользователь совершил действие (username);
    // - когда (timestamp - для простоты условные секунды от начала времён).
    // Внимание: не гарантируется, что timestamp расположены строго по
    // возрастанию, в них может быть любой бардак.
    void click(const string& username, unsigned long long timestamp);

    // По имени пользователя нужно вернуть, сколько всего было кликов
    unsigned long long getClickCount(const string& username) const;

    // Когда был первый клик
    unsigned long long getFirstClick(const string& username) const;

    // Когда был последний клик
    unsigned long long getLastClick(const string& username) const;
};
```

При каждом клике пользователя наш класс уведомляют об этом, вызвав метод click. Поток кликов никак не упорядочен - ни по пользователям, ни по временам событий. Поток кликов нужно как-нибудь учесть, чтобы потом в любой момент уметь отдать по имени пользователя, сколько всего было кликов у данного логина, когда был первый из них и когда последний.

Пример ожидаемого сценария работы:

```
    Tracker t;
    t.click("alice", 1000);
    t.click("bob", 1100);
    t.click("alice", 1001);
    t.click("alice", 1200);
    t.click("alice", 1002);
    cout << t.getClickCount("alice") << endl;
    cout << t.getClickCount("bob") << endl;
    cout << t.getFirstClick("alice") << endl;
    cout << t.getFirstClick("bob") << endl;
    cout << t.getLastClick("alice") << endl;
    cout << t.getLastClick("bob") << endl;
```

В данном примере должна получиться вот такая статистика:

```
4
1
1000
1100
```

1200
1100

# Телеметрия

Напишите класс, который обрабатывает поток событий от устройств и считает по нему очень базовую статистику. Прототип класса:

```
class TelemetryController
{
public:
    // Получить и обработать событие. Параметрами передаются:
    // - device - идентификатор устройства, с которого пришло значение;
    // - value - собственно значение некоторой величины, переданное устройством.
    void handleEvent(const string& device, long value);

    // По идентификатору устройства возвращает,
    // сколько всего значений от него пришло за всё время
    unsigned int getEventsCount(const string& device) const;

    // По идентификатору устройства возвращает
    // минимальное значение за всё время, пришедшее от данного устройства
    long getMinValue(const string& device) const;

    // По идентификатору устройства возвращает
    // максимальное значение за всё время, пришедшее от данного устройства
    long getMaxValue(const string& device) const;
};
```

**Внимание:** в данной задаче не надо хранить в памяти все пришедшие данные в сыром виде. Нужно хранить сразу целевые величины. Считайте, что данных будет настолько много, что в память они не уместятся.

Код для базового тестирования реализации класса:

```
    TelemetryController tc;

    tc.handleEvent("d1", 42);
    tc.handleEvent("d1", -42);
    tc.handleEvent("d2", 100);


    cout << "Events count for d1: " << tc.getEventsCount("d1") << endl;
    cout << "Min value for d1: " << tc.getMinValue("d1") << endl;
    cout << "Max value for d1: " << tc.getMaxValue("d1") << endl;
```

Базовый тест должен вывести:

```
Events count for d1: 2
Min value for d1: -42
Max value for d1: 42
```

# Детектор атак

Напишите класс, который анализирует попытки сетевых подключений к серверу и вычисляет потенциальных нарушителей, которые пытаются атаковать сервер (и которых надо забанить после контрольной проверки на ложные срабатывания анализатора).

Необходимо реализовать следующую логику работы анализатора:

- Анализатор получает уведомление о каждом новом сетевом подключении - с какого внешнего адреса оно выполнено, на какой порт сервера, когда. Анализатор хранит историю подключений в любом виде, удобном для его дальнейшей работы.
- По запросу анализатор проверяет указанный внешний адрес - является он добропорядочным узлом сети или потенциальным нарушителем.
- Для проверки анализатор использует ровно одно правило - если с этого адреса за всё время наблюдения была хотя бы одна попытка массово сканировать порты вашего сервера, то адрес является потенциальным нарушителем.
- Массовым сканированием, выполненным с заданного внешнего адреса, считается последовательность подключений с данного адреса, в которой не менее portLimit разных портов сервера было опрошено за менее timeThreshold миллисекунд.

Информация о сетевых подключениях доступна через следующий интерфейс:

```
class Connection
{
public:
    // Получить внешний адрес, с которого происходит подключение
    string getSource() const;

    // Получить порт вашего сервера, к которому происходит подключение
    unsigned short int getPort() const;

    // Получить время подключения (для простоты - в условных миллисекундах от начала времён)
    unsigned long long getTimestamp() const;
};
```

Необходимо написать класс со следующим прототипом:

```
class IntrusionDetector
{
public:
    // Задать временное окно для анализа (см. описание логики выше)
    void setTimeThreshold(unsigned short int timeThreshold);

    // Задать минимальное количество портов для срабатывания (см. описание логики выше)
    void setPortLimit(unsigned short int portLimit);

    // Вызов этого метода уведомляет анализатор о новом подключении.
    void handleConnection(const Connection& c);

    // Проверить, является ли указанный адрес нарушителем
    bool isIntruder(const string& source) const;
};
```

**Внимание:** ограничения по процессорному времени и оперативной памяти заведомо мягкие, пройдёт любое разумное решение.

**Внимание:** не гарантируется упорядоченность подключений по времени.

Код для базового тестирования реализации класса:

```
    IntrusionDetector id;
    id.setTimeThreshold(5);
    id.setPortLimit(3);
```

```
    id.handleConnection({"evil.com", 21, 100504});
    id.handleConnection({"evil.com", 22, 100501});
    id.handleConnection({"evil.com", 23, 100502});
    id.handleConnection({"evil.com", 24, 100503});
    id.handleConnection({"evil.com", 25, 100500});
    cout << boolalpha << "Checking if evil.com is intruder: " << id.isIntruder("evil.com") <<
endl;

    id.handleConnection({"load.com", 80, 100504});
    id.handleConnection({"load.com", 80, 100501});
    id.handleConnection({"load.com", 80, 100502});
    id.handleConnection({"load.com", 80, 100503});
    id.handleConnection({"load.com", 80, 100500});
    cout << boolalpha << "Checking if load.com is intruder: " << id.isIntruder("load.com") <<
endl;
```

Базовый тест должен вывести:

```
Checking if evil.com is intruder: true
Checking if load.com is intruder: false
```

Во время отладки можете использовать следующую реализацию класса Connection:

```
class Connection
{
protected:
    string source;
    unsigned short int port;
    unsigned long long timestamp;

public:
    Connection(string source, unsigned short int port, unsigned long long timestamp) {
        this->source = source;
        this->port = port;
        this->timestamp = timestamp;
    }
    ~Connection() {}

    string getSource() const {
        return source;
    }

    unsigned short int getPort() const {
        return port;
    }

    unsigned long long getTimestamp() const {
        return timestamp;
    }
};
```