Задача 1(Треугольник)

Задан следующий класс точки на плоскости:

```
#include <cmath>

class Point
{
protected:
    int x;
    int y;

public:
    Point(int x, int y): x(x), y(y) {
    }

    Point operator+(const Point& a) const {
        return Point(x + a.x, y + a.y);
    }

    Point operator-(const Point& a) const {
        return Point(x - a.x, y - a.y);
    }

    double dist() const {
        return sqrt(x * x + y * y);
    }
};
```

Логически точка представлена радиус-вектором, поэтому точки можно складывать и вычитать - соответственные операторы для них уже определены. Также у точки есть метод dist() - получить модуль её радиус-вектора.

Напишите класс треугольника со следующим прототипом:

```
class Triangle
{
public:
    // Создать треугольник из трёх точек
    Triangle(const Point& a, const Point& b, const Point& c);

    // Посчитать и вернуть периметр треугольника
    double perimeter() const;
};
```

Для базового тестирования можете использовать следующий пример:

```
int main()
{
    Point a(0, 0), b(0, 5), c(5, 0);
    Triangle t(a, b, c);
    cout << t.perimeter() << endl;
    return 0;
}
```

Например:

Ввод	Результат
0 0 5 0 0 5	Point #1: (0; 0) Point #2: (5; 0) Point #3: (0; 5) Creating triangle Triangle perimeter: 17.0711

Задача 2(Ломаная)

Реализуйте класс, описывающий ломаную линию на плоскости. Прототип класса:

```
class Polyline {
public:
    // Конструктур и деструктор, если необходимы

    // Добавить очередную точку в ломаную
    void addPoint(double x, double y);

    // Получить количесто точек в ломаной в данный момент
    unsigned int getNumberOfPoints() const;

    // Получить длину ломаной в данный момент
    double getLength() const;

    // Получить x-координату i-ой точки ломаной, точки нумеруются в порядке добавления
    // (Гарантируется, что номер i строго меньше, чем то, вернула ваша getNumberOfPoints())
    double getX(unsigned int i) const;

    // Получить y-координату i-ой точки ломаной, точки нумеруются в порядке добавления
    // (Гарантируется, что номер i строго меньше, чем то, вернула ваша getNumberOfPoints())
    double getY(unsigned int i) const;
};
```

Для тестирования можете использовать следующий пример:

```
int main()
{
    Polyline p;
    p.addPoint(0.0, 0.0);
    p.addPoint(1.0, 0.0);
    p.addPoint(1.0, 1.0);
    cout << "Length: " << p.getLength() << endl;
    cout << "Points:" << endl;
    for(unsigned int i = 0; i < p.getNumberOfPoints(); i++) {
        cout << p.getX(i) << " " << p.getY(i) << endl;
    }
    return 0;
}
```

Например:

Ввод	Результат
5 0 0 1 0 1 1 1 4.5 1.5 4.5	Adding point #0: (0; 0) Adding point #1: (1; 0) Adding point #2: (1; 1) Adding point #3: (1; 4.5) Adding point #4: (1.5; 4.5) Polyline reports these points: 0 0 1 0 1 1 1 4.5 1.5 4.5 Polyline length: 6

Задача 3(Дроби)

Реализуйте простой класс для хранения рациональных чисел в виде дроби с целыми числителем и знаменателем. Реализуйте для данного класса базовые операторы для действий с рациональными числами.

Прототип класса:

```
class Rational {
public:
    // Конструктор дроби, здесь a - числитель, b - знаменатель
    Rational(int a, int b);

    // Реализуйте операторы:
    // - сложения двух дробей
    // - вычитания двух дробей
    // - умножения двух дробей
    // - деления двух дробей
    // - умножения дроби на целое число (должно работать при любом порядке операндов)
};
```

Также реализуйте оператор вывода, который должен выводить дробь в формате: a/b.

Замечания:

- нулевой знаменатель отдельно обрабатывать не требуется, такого случая не будет в тестах
- сокращать дроби не требуется (например, если получилось 4/6, то пусть так и остаётся, преобразовывать в 2/3 не нужно - эти нюансы будут обработаны проверяющим кодом)
- данная реализация, разумеется, получается хрупкая и незавершённая, реальный класс для рациональных чисел будет значительно сложнее

Для тестирования можете использовать следующий пример:

```
int main()
{
    Rational a(1, 2);
    Rational b(1, 3);

    cout << a << endl;
    cout << b << endl;
    cout << a + b << endl;
    cout << a - b << endl;
    cout << a * b << endl;
    cout << a / b << endl;
    cout << 3 * a << endl;
    cout << b * 4 << endl;

    return 0;
}
```

Например:

Ввод	Результат
1 2 1 3 3 4	Creating the first rational r1 from 1 and 2 Creating the second rational r2 from 1 and 3 r1: 1/2 r2: 1/3 r1 + r2: 5/6 r1 - r2: 1/6 r1 * r2: 1/6 r1 / r2: 3/2 r1 * 3: 3/2 4 * r2: 4/3

Задача 4(Транспорт)

Создайте иерархию классов для описания различных транспортных средств. Вам дан следующий базовый интерфейс, который должны реализовать все классы:

```
// Абстрактное транспортное средство
class Vehicle {
public:
    // Может ли ездить по суще
    virtual bool canDrive() const {
        return false;
    }

    // Может ли плавать
    virtual bool canSail() const {
        return false;
    }

    // Может ли летать
    virtual bool canFly() const {
        return false;
    }

    virtual ~Vehicle() {};
};
```

Реализуйте следующие классы:

- Car - автомобиль, является Vehicle, может только ездить по суше
- Ship - корабль, является Vehicle, может только плавать
- Plane - самолёт, является Vehicle, может только летать
- Truck - грузовик, является Car, полностью наследует его поведение

Классы должны соответствующим образом перегрузить методы canDrive, canSail, canFly.

Для тестирования можете использовать следующий пример:

```
int main()
{
    cout << boolalpha;
    Vehicle* v;

    v = new Car();
    cout << "Car can drive: " << v->canDrive() << endl;
    cout << "Car can sail: " << v->canSail() << endl;
    cout << "Car can fly: " << v->canFly() << endl;
    delete v;

    v = new Ship();
    cout << "Ship can drive: " << v->canDrive() << endl;
    cout << "Ship can sail: " << v->canSail() << endl;
    cout << "Ship can fly: " << v->canFly() << endl;
    delete v;

    v = new Plane();
    cout << "Plane can drive: " << v->canDrive() << endl;
    cout << "Plane can sail: " << v->canSail() << endl;
    cout << "Plane can fly: " << v->canFly() << endl;
    delete v;

    Car* c = new Truck();
    cout << "Truck can drive: " << c->canDrive() << endl;
    cout << "Truck can sail: " << c->canSail() << endl;
    cout << "Truck can fly: " << c->canFly() << endl;
```

```
    delete c;

    return 0;
}
```

Например:

Ввод	Результат
4 Car Ship Plane Truck	Testing Car Car can drive: true Car can sail: false Car can fly: false Testing Ship Ship can drive: false Ship can sail: true Ship can fly: false Testing Plane Plane can drive: false Plane can sail: false Plane can fly: true Testing Truck Truck can drive: true Truck can sail: false Truck can fly: false Testing if Truck is Car: true

Задача 5(Диспетчер)

Напишите класс диспетчера, управляющего набором модулей. Диспетчер должен уметь регистрировать модули и вызывать их по запросу.

Модулей будет много и разных, но все они реализуют следующий интерфейс:

```
class Module {
public:
    // Получить имя модуля
    virtual string getName() const = 0;

    // Запустить модуль
    virtual void run() = 0;
};
```

На этапе тестирования можете использовать следующие модули:

```
class ModuleA : public Module {
public:
    string getName() const {
        return "moduleA";
    }

    void run() {
        cout << "ModuleA runs" << endl;
    }
};

class ModuleB : public Module {
public:
    string getName() const {
        return "moduleB";
    }

    void run() {
        cout << "ModuleB runs" << endl;
    }
};
```

Прототип требуемого диспетчера:

```
class Dispatcher {
public:
    // Зарегистрировать переданный модуль
    void registerModule(Module* m);

    // Вызвать метод run для модуля с именем moduleName
    // Если такого модуля не зарегистрировано, ничего не делать (но не падать)
    void runModule(const string moduleName) const;
};
```

Для тестирования своей реализации можете использовать следующий пример:

```
int main()
{
    Module* m1 = new ModuleA();
    Module* m2 = new ModuleB();

    Dispatcher d;
    d.registerModule(m1);
    d.registerModule(m2);
    d.runModule("moduleA");
    d.runModule("moduleB");
    d.runModule("moduleC");

    delete m1;
    delete m2;
```

```
    return 0;
}
```

Например:

Ввод	Результат
2 moduleOne moduleTwo 3 moduleTwo moduleOne moduleX	Registering module: moduleOne Registering module: moduleTwo Running module: moduleTwo ModuleTwo runs Running module: moduleOne ModuleOne runs Running module: moduleX

Задача 6(Обработчик)

Легенда

Вы участвуете в разработке большой клиент-серверной системы. (Пугаться не надо. Реального клиент-серверного кода в задаче нет.) От клиентов приходят различные запросы, сервер на них отвечает. Обработчик каждого типа запросов - это отдельный класс, который может быть достаточно сложно устроен внутри.

Тем не менее, общая процедура обработки запроса всегда одинаковая - выполнить подготовительные действия, проверить права пользователя, обработать запрос, выполнить завершающие действия. Поэтому нужен базовый класс обработчика запроса, который установит интерфейс и реализацию по умолчанию, которыми дальше будут пользоваться все участники разработки.

Постановка задачи

Напишите базовый класс обработчика запроса. Предполагается, что типовая процедура обработки запроса следующая:

```
int main()
{
    // Приём запроса. Извлечение данных пользователя.
    // ... происходит какая-то работа, возникает имя пользователя ...
    string username = "some_user_name";

    // Выполнение какой-то логики для выбора нужного обработчика.
    // Допустим, нужен обработчик SomeHandler. Создаём его.
    Handler* h = new SomeHandler();

    // Далее идёт блок, единый для всех типов обработчиков,
    // ради него и задуман общий интерфейс и базовый класс.

    // Выполнить предварительные действия
    h->preProcess();
    // Проверить, разрешён ли доступ данному пользователю
    if(h->accessAllowed(username)) {
        // Если доступ разрешён, выполнить обработку запроса
        h->run();
    }
    // Выполнить завершающие действия
    h->postProcess();

    // Удалить экземпляр обработчика
    delete h;

    return 0;
};
```

Базовый класс обработчика должен:

- иметь имя `Handler`
- задать технически корректный интерфейс, содержащий публичные методы `void preProcess()`, `void postProcess()`, `void run()`, `bool accessAllowed(string username)`
- предоставить реализации по умолчанию для методов `preProcess`, `postProcess`, `accessAllowed`, при этом по умолчанию методы `preProcess` и `postProcess` должны ничего не делать, а метод `accessAllowed` должен разрешать доступ (возвращать `true`) для всех пользователей
- строго требовать от всех унаследованных классов предоставить реализацию метода `run`

- позволить унаследованным классам использовать свои перегруженные реализации методов `preProcess`, `postProcess`, `accessAllowed`, но не требовать этого (классы без реализации данных методов используют реализацию по умолчанию)
- обеспечить корректное срабатывание цепочек конструкторов и деструкторов для унаследованных классов

Для проверки можете использовать следующие реализации обработчиков:

```
// Должен не работать, так как не предоставляет run
class BadHandler : public Handler {
};

// Должен технически корректно работать
class DummyHandler : public Handler {
public:
    void run() {
        cout << "DummyHandler runs" << endl;
    }
};

// Должен технически корректно работать
// и использовать собственные реализации всех методов
class SmartHandler : public Handler {
public:
    void preProcess() {
        cout << "SmartHandler preprocess" << endl;
    }

    void postProcess() {
        cout << "SmartHandler postprocess" << endl;
    }

    bool accessAllowed(string username) {
        return "admin" == username;
    }

    void run() {
        cout << "SmartHandler runs" << endl;
    }
};

// Должен технически корректно работать
// и очистить память за собой
class HeavyHandler : public Handler {
protected:
    int* data;
public:
    HeavyHandler() {
        data = new int[1000];
    }
    ~HeavyHandler() {
        delete[] data;
    }

    void run() {
        cout << "HeavyHandler runs" << endl;
    }
};
```

Например:

Ввод	Результат
alice 3 HandlerOne HandlerTwo HandlerThree	Running handler HandlerOne for user alice HandlerOne runs Running handler HandlerTwo for user alice HandlerTwo preprocess Access denied HandlerTwo postprocess Running handler HandlerThree for user alice HandlerThree runs

Задача 7(Слова)

Считайте со стандартного потока ввода множество слов. После этого: * устраните дубли, каждое слово должно встречаться строго один раз * выведите все уникальные слова в стандартный поток вывода, отсортировав по длине от самых длинных к самым коротким.

Если попадаются слова одинаковой длины, их взаимный порядок может быть любым.

Формат ввода

В первой строке целое число N - количество слов.

Далее N строк, каждая содержит строго одно слово. Словом считается последовательность символов, которая может включать только малые и большие буквы латинского алфавита.

Формат вывода

X строк, в каждой строке одно уникальное слово, слова отсортированы от длинных к коротким.

Например:

Ввод	Результат
5 foo qwerty aabb qwerty X	qwerty aabb foo X

Задача 8(Медианные значения)

На вход программы приходит множество отсчётов значений, снятых с разных датчиков. Для каждого отсчёта доступны строка (имя датчика) и целое число (собственно значение).

После окончания потока значений на вход программы приходят запросы. Каждый запрос содержит имя датчика (могут встречаться имена, которых не было в множестве значений).

Для каждого запроса нужно вывести медианное значение для данного датчика. Если количество отсчётов на датчике чётное - вывести меньшее из двух чисел в середине диапазона. Если такой датчик не встречался совсем - вывести строку "no data".

Справочно: медианное значение - такое значение, которое стоит в середине массива, если массив упорядочен. Не путать со средним значением. Например, для массива [-10; 1; 2; 100; 100000] медианное значение 2. По условию данной задачи для массива чётной длины берётся меньшее из чисел в середине. То есть, например, для массива [0; 2; 10; 200] нужно брать значение 2.

Формат ввода

В первой строке целое число N - количество отсчётов.

Далее N строк, каждая содержит строку (имя датчика), далее пробел, далее целое число (значение). Имя датчика может содержать малые и большие буквы латинского алфавита, а также цифры.

Далее целое число M - количество запросов.

Далее M строк с именами датчиков, для которых запрашиваются медианные значения.

Формат вывода

M строк, каждая содержит или медианное значение для запрошенного датчика, или строку "no data", если такого датчика не было.

Например:

Ввод	Результат
7 sensor1 80 sensor2 35 sensor1 -4 sensor1 8 sensor2 -1 sensor2 15 sensor2 80 3 sensor1 sensor2 sensor3	8 15 no data

Задача 9(Трекер)

На вход программы приходят зарегистрированные события. Логически каждое событие - утверждение, что некий объект в некий момент времени замечен в некоторой локации. События приходят от независимых наблюдателей, поэтому никак не упорядочены - ни по времени, ни по объектам, ни по локациям.

Считаем, что объекты перемещаются между локациями мгновенно. До первого обнаружения объект находится неизвестно где. После последнего обнаружения объект бесконечно находится в той локации, в которой был замечен в последней.

Гарантируется, что один объект одновременно может быть замечен только в одной локации.

То есть если некий объект:

- первый раз был замечен в момент времени A в локации X,
- затем в момент времени B замечен в локации Y,
- последний раз был замечен в момент времени C в локации Z,

то до момента A он находился неизвестно где, во время [A, B) он находился в локации X, во время [B, C) он находился в локации Y, далее от момента времени C и до бесконечности он находился в локации Z.

Необходимо обработать поток событий и уметь отвечать на запросы, где что сейчас находится.

Формат ввода

В первой строке целое число N - количество событий.

Далее N строк, каждая строка содержит:

- uuid - идентификатор объекта, строка, может содержать малые буквы латинского алфавита, а также символ '-'
- location - название локации, строка, может содержать малые и большие буквы латинского алфавита, а также цифры
- timestamp - метка времени, целое число без знака

Далее в отдельной строке целое число M - количество запросов.

Далее M строк с запросами. Каждый запрос может иметь один из двух видов:

- или "object uuid ts" - запрос данных по объекту, здесь object это фиксированное ключевое слово, uuid содержит идентификатор объект, ts содержит метку времени
- или "location name ts" - запрос данных по локации, здесь location это фиксированное ключевое слово, name содержит название локации, ts содержит метку времени

Формат вывода

Для каждого запроса вида "object uuid ts" нужно вывести:

- или строку с именем локации, в которой находился объект в данный момент,
- или строку "no data", если нет данных по данному объекту в данный момент времени.

Для каждого запроса вида "location name ts" нужно вывести:

- или строку с uuid-ами объектов, которые находились в данной локации в данный момент (uuid через пробел, упорядоченные по алфавиту),

- или пустую строку, если в данной локации пусто в данный момент,
- или строку "no data", если вообще нет данных по данной локации.

Например:

Ввод	Результат
6 cdded834-64ec-11e9-a923-1681be663d3e base1 100 cdded834-64ec-11e9-a923-1681be663d3e base2 150 cdded834-64ec-11e9-a923-1681be663d3e base3 125 cdded834-64ec-11e9-a923-1681be663d3e base4 800 cddedb7c-64ec-11e9-a923-1681be663d3e base2 120 cddedb7c-64ec-11e9-a923-1681be663d3e base1 400 12 object cdded834-64ec-11e9-a923-1681be663d3e 50 object cdded834-64ec-11e9-a923-1681be663d3e 130 object cdded834-64ec-11e9-a923-1681be663d3e 1000 object cddedb7c-64ec-11e9-a923-1681be663d3e 110 object cddedb7c-64ec-11e9-a923-1681be663d3e 120 object cddedb7c-64ec-11e9-a923-1681be663d3e 180 object aaaaaaaa-64ec-11e9-a923-1681be663d3e 200 location base2 1000 location base2 120 location base2 121 location base2 150 location base2 151	no data base3 base4 no data base2 base2 no data cddedb7c-64ec-11e9-a923-1681be663d3e cddedb7c-64ec-11e9-a923-1681be663d3e cdded834-64ec-11e9-a923-1681be663d3e cddedb7c-64ec-11e9-a923-1681be663d3e cdded834-64ec-11e9-a923-1681be663d3e cd