Анализ нагрузки

Легенда

Пишем нечто смутно похожее на анализатор нагрузки на многопроцессорную систему.

Считаем, что в системе одновременно выполняется множество процессов, каждый из которых выполняется строго на одном ядре процессора, нагрузку от каждого можно характеризовать одной цифрой с логическим смыслом "вычислительная сложность **задачи**" *(в реальной жизни всё гораздо сложнее, но нам для примера пойдёт)*.

Предполагаем, что наш анализатор просыпается раз в N миллисекунд, просматривает текущую ситуацию с задачами на выполнение, считает по ней нагрузку на ядра процессора. До следующего просыпания и анализа планировщик считает картину статичной *(что, разумеется, не соответствует действительности, ну да и ладно)*.

Постановка задачи

Каждый процесс в системе описывается вот таким классом:

```
class Task
{
protected:
    int cpuNum;
    int size;

public:
    Task(int cpuNum, int size) {
        this->cpuNum = cpuNum;
        this->size = size;
    }

    // На каком ядре процессора выполняется задача
    int getCPU() const {
        return cpuNum;
    }

    // Оценка сложности задачи (в попугаях)
    int getSize() const {
        return size;
    }
};
```

Напишите класс анализатора вот с таким прототипом:

```
class Analyzer
{
public:
    // Создать анализатор для системы с numCores ядер
    Analyzer(int numCores);

    // Проанализировать текущие задачи
    void analyze(const vector<Task>& tasks);

    // Сообщить общую нагрузку на заданное ядро
    int getLoadForCPU(int cpuNum);
};
```

Метод analyze получает на вход вектор с описанием всех текущих задач в системе и выполняет его анализ. После выполненного анализа метод getLoadForCPU по номеру ядра процессора сообщает текущую нагрузку на него (считается как сумма сложностей всех процессов на данном ядре).

Пример ожидаемого сценария работы всей конструкции:

```
int numberOfCores = 4;
vector<Task> data = { {0, 1}, {1, 10}, {0, 6}, {2, 12}, {3, 5} };
Analyzer a(numberOfCores);
a.analyze(data);
for(innt i = 0; i < numberOfCores; i++)
    cout << a.getLoadForCPU(i) << endl;
```

Должно напечатать текущую нагрузку на четырёх ядрах:

```
7
10
12
5
```

Турнир

Задача может быть решена как с STL, так и без него - выбор за вами.

Проходит турнир, участвует множество игроков. Напишите класс, который получает результаты, сохраняет их в каком-либо удобном для себя виде и умеет в дальнейшем отвечать на вопрос, сколько баллов у игрока на заданном месте в итоговой таблице. (Как зовут игрока - в рамках данной задачи неважно.)

Прототип класса:

```
class ResultsTable
{
public:
    // Зарегистрировать новый результат,
    // нас волнуют только баллы, имена пользователей не важны
    void addResult(unsigned int score);

    // Получить минимальный балл из всех результатов за всё время
    unsigned int getMinScore() const;

    // Получить, сколько баллов у игрока на заданном месте.
    // Внимание: места нумеруются так, как это принято на турнирах, то есть
    // лучший результат - 1-ое место, за ним 2-ое место и т.д.
    unsigned int getScoreForPosition(unsigned int positionNumber) const;
};
```

Внимание: в прототипе класса есть модификаторы const на методах. Не забывайте про них!

Код для базового тестирования реализации класса:

```
    ResultsTable t;

    t.addResult(30);
    t.addResult(85);
    t.addResult(12);
    t.addResult(31);

    cout << "1st place score: " << t.getScoreForPosition(1) << endl;
    cout << "2nd place score: " << t.getScoreForPosition(2) << endl;
    cout << "3rd place score: " << t.getScoreForPosition(3) << endl;
    cout << "Min score during the tournament: " << t.getMinScore() << endl;
```

Базовый тест должен вывести:

```
1st place score: 85
2nd place score: 31
3rd place score: 30
Min score during the tournament: 12
```

Если у нескольких игроков одинаковые результаты, то они делят идущие подряд несколько мест. Например, если игроков всего 3, и у всех по 42 балла, то getScoreForPosition для первых трёх мест должен вернуть 42.

ФИО

Пишем класс, который будет "STL-friendly" - обеспечиваем, чтобы базовые контейнеры и алгоритмы STL могли работать с нашим классом. (В задаче есть пара небольших подводных камней. Это нормально, так задумано.)

Напишите класс, описывающий человека с ФИО:

```
class Person
{
public:
    // Создать человека с ФИО
    Person(string surname, string name, string middleName);
};
```

Реализуйте операторы >> (ввод данных человека), << (вывод данных человека), < (сравнение людей по ФИО по словарю). Также реализуйте все вспомогательные методы, которые потребуются для работы описанного.

Оператор ввода >> должен считывать и заполнять ФИО. Формат ввода: фамилия, затем имя, затем отчество.

Оператор вывода << должен выводить ФИО. Формат вывода: фамилия, затем имя, затем отчество.

Оператор < сравнивает сначала фамилии (как строки по словарю), при равенстве фамилий - имена, при равенстве имён - отчества.

Приблизительный код для тестирования реализованного класса (не забудьте include-ы для vector и sort):

```
    cout << "Testing I/O" << endl;
    Person p;
    cin >> p;
    cout << p << endl;

    cout << "Testing sorting" << endl;
    vector<Person> people;
    people.push_back(Person("Ivanov", "Ivan", "Ivanovich"));
    people.push_back(Person("Petrov", "Petr", "Petrovich"));
    people.push_back(Person("Ivanov", "Ivan", "Petrovich"));
    people.push_back(Person("Ivanov", "Petr", "Ivanovich"));

    sort(people.begin(), people.end());
    for(vector<Person>::const_iterator it = people.begin(); it < people.end(); it++) {
        cout << *it << endl;
    }
```

Для входных данных (с клавиатуры):

```
A B C
```

Должно вывести:

```
Testing I/O
A B C
Testing sorting
Ivanov Ivan Ivanovich
Ivanov Ivan Petrovich
Ivanov Petr Ivanovich
Petrov Petr Petrovich
```