Задача 1

Напишите класс `CalendarData`, в котором определены:
- конструктор с параметрами (день, месяц, год), каждое из которых представляет из себя `int`;
- деструктор, который автоматически при уничтожении экземпляра объекта выводит на поток `std::cout` полученные в констукторе "день.мексяц.год" (через точку без пробелов).

Других методов, кроме конструктора и деструктора, описывать не требуется.

Формат входных данных
Считывать с клавиатуры ничего не требуется. Дата попадает в объект класса через конструктор с параметрами.

Формат выходных данных
Три числа через точку на поток `std::cout`.

Примеры

-> CalendarData data(5, 8, 2005);
--
<- 5.8.2005

-> CalendarData data(15, 12, 2012);
--
<- 15.12.2012

-> CalendarData data(3, 3, 333);
--
<- 3.3.333

Задача 2

Реализовать класс `FrequencyTree` частотного словаря, используя двоичное дерево поиска.

Класс должен иметь:

1. Конструктор по умолчанию, без параметров.
2. Метод `void addValue(int)` - принимает число и добавляет его в дерево, при этом для числа, добавляемого впервые, частота - 1, для вторично добавляемого числа частота увеличивается на 1.
3. Метод `void printValues()` - печатает содержимое словаря в порядке возрастания величины числа (см. Формат выходных данных).

Реализовать нужно только класс, main-функцию реализовывать не нужно.

Допускается реализация рядом с основным классом служебного класса или структуры `TreeNode` для узла в дереве.

Формат входных данных
Ничего считывать с клавиатуры не требуется.
Будет вызван метод `ftree.addValue(value);` для добавления каждого числа.

Формат выходных данных
Метод `ftree.printValues();` выводит на экран содержимое дерева в порядке возрастания значений, сохранённых в дереве, по одному элементу на строку.
В каждой строке выводите значение элемента, затем, через пробел - частота, то есть сколько раз он встречается в исходной последовательности.

Примеры

```
-> FrequencyTree ftree;
-> ftree.addValue(4);
-> ftree.addValue(4);
-> ftree.printValues();
--
<- 4 2
```

```
-> FrequencyTree ftree;
-> ftree.addValue(2);
-> ftree.addValue(2);
-> ftree.addValue(2);
-> ftree.addValue(3);
```

```
-> ftree.addValue(3);
-> ftree.printValues();
--
<- 2 3
<- 3 2
```

Задача 3

Написать классы с заданными интерфейсами, которые реализуют часть логики многопользовательской игры: Предмет интвентаря, Инвентарь, Игрок, Группа игроков.

```
/**
 * Класс предмета в инвентаре
 */
class Item {
 public:
    /**
     * Конструктор
     * @param name название
     * @param weight вес
     * @param price цена
     */
    Item(const std::string& name, unsigned weight, unsigned price);

    /**
     * Получить название предмета
     * @return название
     */
    const std::string& get_name() const;

    /**
     * Получть вес предмета
     * @return вес
     */
    unsigned get_weight() const;

    /**
     * Узнать цену предмета
     * @return цена
     */
    unsigned get_price() const;

    /**
     * Напечатать информацию о предмете в формате ": название вес цена"
     * @param os поток вывода
     */
    void print(std::ostream& os) const;
};

/**
```

```
 * Класс инвентаря.
 */
class Inventory {
 public:
    /**
     * Конструктор
     * @param size вместимость (максимальный вес всех предметов) равен силе
игрока
     */
    explicit Inventory(unsigned size);

    /**
     * Положить предмет в инвентарь, если для него есть место
     * @param item предмет
     * @return true в случае успеха
     */
    bool put(const Item& item);

    /**
     * Распечатать все предметы в порядке получения игроком
     * @param os поток вывода
     */
    void print(std::ostream& os) const;
};

/**
 * Класс, описывает отдельного игрока
 */
class Player {
 public:
    /*
     * Конструктор
     * @param name имя
     * @param strength сила
     */
    Player(const std::string& name, unsigned strength);

    /**
     * Узнать имя игрока
     * @return имя
     */
    const std::string& get_name() const;

    /**
     * Взять предмет
```

```
     * @param item предмет
     * @return true если он поместился в инвентарь
     */
    bool take(const Item& item);

    /**
     * Распечатать на первой строке имя игрока, а на следующих содержимое его
инвентаря
     * @param os поток вывода
     */
    void print(std::ostream& os) const;
};

/**
 * Класс, описывающий группу игроков
 */
class Party {
 public:
    /*
     * Добавить игрока в группу, если игрока с таким имененм в ней ещё нет
     * @param player игрок
     * @return true если игрок был успешно добавлен
     */
    bool add(const Player& player);

    /*
     * Дать предмет игроку
     * @param player_name имя игрока
     * @param item предмет
     * @return true если игрок успешно положил его в инвентарь
     */
    bool give(const std::string& player_name, const Item& item);

    /*
     * Распечатаь всех игроков в алфавитном порядке
     * @param os поток вывода
     */
    void print(std::ostream& os) const;
};
```

Нужно дописать классы и реализацию методов.
Остальная игра уже написана и публичный интерфейс менять нельзя.

Пример использования

```
int main()
{
    Party p;
    p.add(Player("Anti-Mage", 15));
    p.add(Player("Razor", 18));
    p.give("Razor", Item("Necronomicon", 1, 5));
    p.give("Anti-Mage", Item("Refresher_Orb", 2, 2));
    p.print(cout);

    return 0;
}
```

Результат выполнения

```
Anti-Mage
:Refresher_Orb 2 2
Razor
:Necronomicon 1 5
```

Задача 4

Написать класс `MyClass` с инкапсулированным `std::vector` вектором структур (пар). В классе должны быть реализованы два метода:

- `addElement()` - добавить новую структуру в массив. Если точно такая же уже есть - не добавлять, и ничего не делать.
- `printStructures()` - распечатать содержимое элементов массива в формате "число строка" (через пробел).

Конструктор и деструктор в данном классе реализовывать *не требуется*.

Структура-элемент вектора должна содержать в себе поля `int` и `std::string`.
Если вы будете реализовывать собственную структуру, допустимо определить для неё `operator==`.
Допустимо (и даже рекомендуется) использовать в качестве структуры *стандартный* "микроконтейнер"
`std::pair<int, std::string>` и функцию `std::make_pair(x, s)`.

Обратите внимание, что структуры считаются равными друг другу, если равны и числа, и строки.

Формат выходных данных
Вывод в стандартный поток вывода происходит в результате вызова `obj.printStructures()`.
В результате выводятся все хранимые структуры, каждая в новой строке, при этом число и строка разделены пробелом.

Пример
```
-> MyClass obj1;
-> obj1.addElement(1, "hello");
-> obj1.addElement(2, "hi");
-> obj1.addElement(2, "hi");
-> obj1.printStructures();
--
<- 1 hello
<- 2 hi
```

Задача 5

Добавьте в предыдущей задаче (Класс с массивом структур) перегруженный оператор суммы += , который соединяет массив структур класса слева с массивом стуктур класса справа. Если элементы второго класса совпадают с элементами первого - они не добавляются.

Формат входных данных
Число и строка, разделенные пробелом

Формат выходных данных
Элементы суммарного массива, где на каждой строчке выводится число и строка, разделенные пробелом

Пример

```
-> MyClass obj1;
-> obj1.addElement(1, "hi");
-> obj1.addElement(2, "hello");
-> obj1.addElement(2, "hello");
-> MyClass obj2;
-> obj2.addElement(3, "hey");
-> obj2.addElement(4, "pop");
-> obj2.addElement(2, "hello");
-> obj2.addElement(5, "hello");
-> obj1 += obj2;
-> obj1.printStructures();
--
<- 1 hi
<- 2 hello
<- 3 hey
<- 4 pop
<- 5 hello
```