Решение задач можно присылать (предпочтительнее ссылка на github, и т. п.) на vkonovodov@gmail.com. Бонусные баллы за решения задач могут быть поставлены только нескольким первым приславшим правильное решение. Решение второй задачи должно быть оформлено в виде компилирующегося кода.

Задача 1

Приведите пример такого STL-контейнера `v`, для которого конструкции

```
for (auto&& x : v ) {...}
```

и

```
for (auto& x : v ) {...}
```

отличаются по выводу типа или компилируемости. Поясните, почему.

Задача 2

Реализуйте функцию pretty-printer'а для форматированного представления объектов. Необходимо поддержать: целочисленные типы, `std::string`, `std::pair`, `std::vector`, `std::set`, `std::tuple` и их композиции.

- `std::set` представляется в виде `{ element, ..., element }`,
- `std::vector` представляется в виде `[ element, ..., element ]`,
- `std::pair` представляется в виде `( item, item )`,
- `std::tuple` представляется в виде `( element, ..., element )`.

Требуемый интерфейс:

```
struct Printer {
    std::string str() const; // возвращает строковое представление
    // далее перегруженные/шаблонные функции вида:
    Printer& format(/*...*/);
}

template<typename T>
std::string format(const T& t) {
    return Printer().format(t).str();
}
```

Метод `format` должен поддерживать цепные вызовы, дающие в результате конкатенацию соответствующих строк.

Примеры:

```
int main() {
    std::tuple<std::string, int, int> t = {"xyz", 1, 2};
    std::vector<std::pair<int, int> > v = {{1, 4}, {5, 6}};

    std::string s1 = Printer().format("  vector:  ").format(v).str();
    // "  vector:  [ (1, 4), (5, 6) ]"
    std::string s2 = Printer().format(t).format(" ! ").format(0).str();
    // "( xyz, 1, 2 ) ! 0"
}
```

Советы:

- Воспользуйтесь `std::stringstream`.
- Постарайтесь избежать дублирования кода.