```
====== array.h ========

#pragma once

struct Array
{
        inline Array(size_t size)
                : data_(new int[size])
                , size_ (size)
        {
                ++instances_;
        }

        ~Array()
        {
                --instances_;
        }

        ~~static void swap_arrays(Array &a, Array &b~~);

        ....

private:
        Array(Array const &);
        Array & operator=(Array const&);

private:
        int *           data_;
        size_t          size_;
        static int  instances_;
};

inline void f(){}

extern int global;
```

```
====== array.cpp =======

#include "array.h"

~~static int swaps = 0;~~
int Array::instances_ = 0;

static void swap(int & a, int & b)
{
        int tmp = a;
        a = b;
        b = tmp;
        ++swaps;
}

static void swap(int *& a, int *& b)
{
        int * tmp = a;
        a = b;
        b = tmp;
        ++swaps;
}

void Array::swap_arrays(Array &a, Array &b
{
        swap(a.data_, b.data_);
        swap(a.size_,  b.size_);
}

int global = 0;
```

```
struct Singleton
{

static Singleton & getinstance() {
        static Singleton s;
        return s;
}

private:
        Singleton(){}
        Singleton(Singleton const& a);
        Singleton operator=(Singleton const& a);
        ~Singleton(){}
};

inline void f() {
    static int k = 0;
}

static void g() {
    static int m = 0;
}
```