Question 1：

```
#include <iostream>
using namespace std;

template <typename T>
class shared_ptr{
public:
    //构造函数
    shared_ptr(T* p = nullptr): _ptr(p), _count(new size_t) {
        if (p)
            *_count = 1;
        else
            *_count = 0;
    }

    //析构函数
    ~shared_ptr() {
        if (--(*_count) == 0) {
            delete _ptr;
            delete _count;
        }

    //赋值（重载）
    shared_ptr& operator=(const shared_ptr & value) {
            if (_ptr == value._ptr) { return this; }
            release();
            _ptr = value._ptr;
            _reference_count = value._count;
            (*_count)++;
            return *this;
    }

}
private:
    T* _ptr;
    size_t* _count;
    //释放资源函数
    void release() {
        if (_ptr) {
            (*_count)--;
            if ((*_count) == 0) {
                delete _ptr;
                delete _count;
            }
        }
    }
};

int main()
{
    shared_ptr<char> t1(new char('a'));
    shared_ptr<float> t2(new float('1.2'));
    shared_ptr<int> t3(new int('3'));
    shared_ptr<int> t4;
```

```
    t4 = t1;
    t4 = t2;
    t4 = t3;
}
```

Question 2:

```
struct Person {
    float x;//横坐标
    float y;//纵坐标
    float weight;//体重
};

class priority_queue {
    Person data[128]; //存储元素的数组
    int count = 0; // 当前队列中的元素个数
public:
    bool empty() {
        return count == 0;
    }
    Person top() {
        if (!empty()) {
            return data[1];
        }
    }

    void push(Person p) {
        count++;
        data[count] = p;
        swim(count);
    }

    Person pop() {
        Person p = data[count];
        exch(1, count);
        data[count].weight = -1;
        count--;
        sink(1);
        return p;
    }
    //上浮第k个元素
    void swim(int k) {
        while (k > 1 && less(parent(k), k)) {
            exch(parent(k), k);
            k = parent(k);
        }
    }
    //下沉第k个元素
    void sink(int k) {
        while (left(k) <= count) {
            int older = left(k);
            if (right(k) <= count && less(older, right(k))) {
                older = right(k);
            }
            if (less(older, k)) break;
```

```
            exch(k, older);
            k = older;
        }
    }

    int parent(int root) {
        return root / 2;
    }
    int left(int root) {
        return root * 2;
    }
    int right(int root) {
        return root * 2 + 1;
    }

    // 判断data[i] 是否比data[j]小
    bool less(int i, int j) {
        return data[i].weight > data[j].weight;
    }
    //交换数组的两个元素
    void exch(int i, int j) {
        Person temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }
};

int main()
{
    priority_queue pq;
    for (int i = 0; i < 10; i++) {
        struct Person p = { 1,2,10-i };
        pq.push(p);
    }
    for (int i = 0; i < 10; i++) {
        Person bb = pq.pop();
        cout << bb.weight << endl;
    }
    return 0;
}
```