```
#ifndef XHASHMAP_H
#define XHASHMAP_H
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <memory.h>
using namespace std;

#include "hash/IMap.h"

template<class K, class V>
class XHashMap : public IMap<K, V>{
protected:
    int (*hashCode)(K&,int); //hasCode(K key, int tableSize): tableSize means capacity
    bool (*keyEqual)(K&,K&);  //keyEqual(K& lhs, K& rhs): test if lhs == rhs
    bool (*valueEqual)(V&,V&); //valueEqual(V& lhs, V& rhs): test if lhs == rhs
    void (*deleteKeys)(XHashMap<K,V>*); //deleteKeys(XHashMap<K,V>* pMap): delete all
    keys stored in pMap
    void (*deleteValues)(XHashMap<K,V>*); //deleteValues(XHashMap<K,V>* pMap): delete
    all values stored in pMap

public:
    XHashMap(
            int (*hashCode)(K&,int),
            bool (*valueEqual)(V&, V&)=0,
            void (*deleteValues)(XHashMap<K,V>*)=0,
            bool (*keyEqual)(K&, K&)=0,
            void (*deleteKeys)(XHashMap<K,V>*)=0);
};

template<class K, class V>
XHashMap<K,V>::XHashMap(
        int (*hashCode)(K&,int),
        bool (*valueEqual)(V& lhs, V& rhs),
        void (*deleteValues)(XHashMap<K,V>*),
        bool (*keyEqual)(K& lhs, K& rhs),
        void (*deleteKeys)(XHashMap<K,V>* pMap) ){

    this->hashCode = hashCode;
    this->valueEqual = valueEqual;
    this->deleteValues = deleteValues;
    this->keyEqual = keyEqual;
    this->deleteKeys = deleteKeys;

}

#endif /* XHASHMAP_H */
```