```c
/* OrderVector.h */
#ifndef _ORDER_VECTOR_H
#define _ORDER_VECTOR_H

#define INITALLOC 4

//0--代表比较相同，1--代表dataAddr<keyAddr，-1--代表dataAddr>keyAddr
typedef int  VectorCmp(const void *keyAddr, const void *dataAddr);
typedef void VectorFree(void *);
typedef void VectorTraverseOp(void *, void *);

typedef struct vector
{
    void *elems; //存放有序表元素的首地址
    int  elemSize; //有序表每个元素占用字节数
    int  size; //有序表目前使用的元素数量
    int  capacity; //有序表目前分配的元素数量
    int  fSupportGrow; //有序表是否支持扩容
    VectorCmp *cmpFn;
    VectorFree *freeFn;
}VECTOR;

//新建线性表
void VectorNew(VECTOR *v, int elemSize, int capacity, int fSupportGrow, VectorCmp *cmpFn, VectorFree *freeFn);
//销毁线性表
void VectorDispose(VECTOR *v);
//判断线性表是否为空
int VectorEmpty(VECTOR *v);
//判断线性表是否已满
int VectorFull(VECTOR *v);
//线性表元素数量
int VectorSize(VECTOR *v);
//清空线性表元素
void VectorMakeEmpty(VECTOR *v);
//根据位置查找元素，返回值为元素地址
void *VectorGetByPos(VECTOR *v, int pos);
//根据值查找元素，way!=0线性查找，way=0二分查找，返回值为不小于该元素的最小位置
int VectorSearch(VECTOR *v, const void *e, int way);
//判断关键码是否在向量的第pos个置位，返回值：0--不在，!0--存在
int VectorFind(VECTOR *v, int pos, const void *e);
//根据位置插入元素（慎用，可能会破坏有序性），返回值：!0--插入失败，0--插入成功
int VectorInsertByPos(VECTOR *v, const void *e, int pos);
//插入元素，返回值：!0--插入失败，0--插入成功
int VectorInsert(VECTOR *v, const void *e);
//根据位置删除元素，返回值：!0--删除失败，0--删除成功
int VectorRemoveByPos(VECTOR *v, int pos);
//删除元素，返回值：!0--删除失败，0--删除成功
int VectorRemove(VECTOR *v, void *e);
//根据位置删除元素（无需深度删除），返回值：!0--删除失败，0--删除成功
int VectorRemoveByPosU(VECTOR *v, int pos);
//删除元素（无需深度删除），返回值：!0--删除失败，0--删除成功
int VectorRemoveU(VECTOR *v, void *e);
//遍历线性表
void VectorTraverse(VECTOR *v, VectorTraverseOp *traverseOpFn, void *outData);
//交换两个表的元素，返回值：!0--交换失败，0--删除成功
int VectorSwap(VECTOR *v, VECTOR *u, int rankV, int rankU);
#endif
```