```
//-----------------------------------------------------------------------------------------------------------------------
///class Hash defines the interface for Separate Chaining and Open Addressing Hash
//implementations
//-----------------------------------------------------------------------------------------------------------------------
class Hash {
protected:
    unsigned int Index(int size,string key);        //Map the key to an integer index
public:
    virtual void Insert(string key)=0;              //Insert a key
    virtual void Remove(string key)=0;              //Remove a key
    virtual bool IsMember(string key)=0;            //Is key in the table
    virtual bool IsFull(void)=0;                    //Is the table full?
    virtual bool IsEmpty(void)=0;                   //Is the table empty?
    virtual int KeyIndex(string key)=0;             //Return the integer index for key
};
unsigned int Hash::Index(int size,string key)
{   unsigned int ndx=0;
    for (int i=0;i<key.length();i++) ndx=(ndx<<4)^(int)key[i];
    return ndx% size;
}
```

```
/------------------------------------------------------------------------------------------------------------------------
//Class OHash defines the interface for open addressing
/------------------------------------------------------------------------------------------------------------------------
enum State {available,occupied,removed};
//-------------------------------------------------------------------------
//implementation
//-------------------------------------------------------------------------
class OHash : public Hash {
    int size;                                   //Number of available entries in the table
    int count;                                  //Number of occupied entries in the table
    //-------------------------------------------------------------------
    //struct Entry contains the attributes of an entry in an open addressing hash table
    //-------------------------------------------------------------------
    struct Entry {
        State state;
        string key;
        Entry():state(available),key("") {};    //Default constructor for the array allocation
        bool Occupied(void);                    //Determine if the entry is occupied
        bool InUse(void);                       //Determine if the entry is either occupied or
                                                //removed
    };
    Entry* L;                                   //Points to an array of entries
public:
                                                //Constructor
    OHash(int sz=37):size(sz),count(0){L=new Entry[size];}
    ~OHash();                                   //Destructor
    void Insert(string key)                     //Inserts a key into the table
    {    if (IsFull()) throw OHashFullException();
        unsigned int ndx;
        for(ndx=Index(size,key);L[ndx].Occupied();ndx=(ndx+1)% size) {
            if (key==L[ndx].key) return;
        }
        L[ndx].state=occupied;
        L[ndx].key=key;
        count++;
    }
    void Remove(string key);                    //Removes a key from the table
    bool IsMember(string key);                  //Determines if parameter key is in the table
    bool IsFull(void);                          //Determines if the table is full.
    bool IsEmpty(void);                         //Determines if the table is empty.
    int KeyIndex(string key);                   //Returns the index of parameter key
    class OHashFullException{};
};
```