```
//-----------------------------------------------------------------------------------------------------------------------
///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 List defines the attributes of a list of strings.
//The list is implemented as an array of strings stored in lexicographic (alphabetic) order.
//The list contains a sentinel. L[0]="";
//------------------------------------------------------------------------------------------------------------------------
class List {
      int size;                                   //Number of available elements
      int count;                                  //Number of occupied elements
      string* L;                                  //Points to an array of strings
public:
      List(int sz=10) :size(sz),count(0)          //Constructor
      {     L=new string[size];
            L[0]="";
      }
      ~List();                                    //Destructor
      class ListFullException {};                 //Thrown when the list is full
      class ListEmptyException {};                //Thrown when the list is empty
      bool IsFull(void);                          //Determines if the list is full
      bool IsEmpty(void);                         //Determines if the list is empty
      int Insert(string key)                      //Inserts a key in the list
      {     if (IsMember(key)) return 0;
            if (IsFull()) throw ListFullException();
            int i;
            for (i=++count;key<L[i-1];i--) L[i]=L[i-1];
            L[i]=key;
            return 1;
      }
      int Remove(string key)                      //Removes a key from the list
      {     int i=Index(key);
            if (i==0) return 0;
            for(;i<count;i++) L[i]=L[i+1];
            count--;
            return -1;
      }
      bool IsMember(string key);                  //Determines if the key is a member of the list
      int Index(string key);                      //Returns the index of the key
                                                  //Implemented as a binary search
                                                  //or zero if the key is not a member of the list
      int Count(void);                            //Returns the number of keys on the list
};
```

```
/*-------------------------------------------------------------------------------------------------------------------
//class SHash defines the attributes of a Hash Table where collisions are resolved by putting them on
//a separate chain.  A separate chain is implement as a list.
//-------------------------------------------------------------------------------------------------------------------*/
class SHash : public Hash {
     int size;                                  //Number of available entries in the table
     int count;                                 //Number of occupied entries in the table
     List* L;                                   //Points to a array of collision resolution lists.
public:
     SHash(int sz=37)                           //Constructor
          :size(sz),count(0){ L=new List[size];}
     ~SHash();                                  //Destructor
     void Insert(string key);                   //Inserts a key into the table
     {    unsigned int ndx=Index(size,key);
          int n=L[ndx].Count();
          count+=L[ndx].Insert(key);
     }
     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 the collision resolution
                                                //chain for parameter key
     int CollisionLength(string key);           //Returns the length of the collision resolution
                                                //chain for parameter key
};
```