```
1   //------------------------------------------------------------------------
2   #ifndef CHHASHTABLEH
3   #define CHHASHTABLEH
4   //------------------------------------------------------------------------
5
6   #ifdef _MSC_VER
7   #pragma warning(disable: 4290 ) // Suppress warning: C++ Exception Specification ignored
8   #endif
9
10  #include <string>
11  #include "ObjectAllocator.h"
12  #include "support.h"
13
14  // client-provided hash function: takes a key and table size,
15  // returns an index in the table.
16  typedef unsigned (*HASHFUNC)(const char *, unsigned);
17
18  // Max length of our "string" keys
19  const unsigned MAX_KEYLEN = 10;
20
21  class HashTableException
22  {
23      private:
24          int error_code_;
25          std::string message_;
26
27      public:
28          HashTableException(int ErrCode, const std::string& Message) :
29              error_code_(ErrCode), message_(Message) {};
30
```

```
        virtual ~HashTableException() {
        }

        virtual int code(void) const {
            return error_code_;
        }

        virtual const char *what(void) const {
            return message_.c_str();
        }
        enum HASHTABLE_EXCEPTION {E_ITEM_NOT_FOUND, E_DUPLICATE, E_NO_MEMORY};
};


// HashTable statistical info
struct HTStats
{
    HTStats(void) : Count_(0), TableSize_(0), Probes_(0), Expansions_(0),
    HashFunc_(0) {};
    unsigned Count_;      // Number of elements in the table
    unsigned TableSize_;  // Size of the table (total slots)
    unsigned Probes_;     // Number of probes performed
    unsigned Expansions_; // Number of times the table grew
    HASHFUNC HashFunc_;   // Pointer to primary hash function
};

template <typename T>
class ChHashTable
{
    public:

        typedef void (*FREEPROC)(T); // client-provided free proc (we own the data)

        struct HTConfig
        {
            HTConfig(unsigned InitialTableSize,
                    HASHFUNC HashFunc,
                    double MaxLoadFactor = 3.0,
                    double GrowthFactor = 2.0,
                    FREEPROC FreeProc = 0) :

                InitialTableSize_(InitialTableSize),
                HashFunc_(HashFunc),
                MaxLoadFactor_(MaxLoadFactor),
                GrowthFactor_(GrowthFactor),
                FreeProc_(FreeProc) {}

            unsigned InitialTableSize_;
            HASHFUNC HashFunc_;
            double MaxLoadFactor_;
            double GrowthFactor_;
            FREEPROC FreeProc_;
```

```
        };

        // Nodes that will hold the key/data pairs
        struct ChHTNode
        {
            char Key[MAX_KEYLEN]; // Key is a string
            T Data;               // Client data
            ChHTNode *Next;
            ChHTNode(const T& data) : Data(data) {}; // constructor
        };

        // Each list has a special head pointer
        struct ChHTHeadNode
        {
            ChHTNode *Nodes;
            ChHTHeadNode(void) : Nodes(0), Count(0) {};
            int Count; // For testing
        };

        ChHashTable(const HTConfig& Config, ObjectAllocator* allocator = 0);
        ~ChHashTable();

        // Insert a key/data pair into table. Throws an exception if the
        // insertion is unsuccessful.
        void insert(const char *Key, const T& Data) throw(HashTableException);

        // Delete an item by key. Throws an exception if the key doesn't exist.
        // Compacts the table by moving key/data pairs, if necessary
        void remove(const char *Key) throw(HashTableException);

        // Find and return data by key (throws exception if not found)
        const T& find(const char *Key) const throw(HashTableException);

        // Removes all items from the table (Doesn't deallocate table)
        void clear(void);

        // Allow the client to peer into the data
        HTStats GetStats(void) const;
        const ChHTHeadNode *GetTable(void) const;

    private:
        HTConfig config_;
        mutable HTStats stats_{};
        ObjectAllocator* oa_;
        ChHTHeadNode* table_;

        ChHTNode* make_node(const char* Key, const T& Data);
        void delete_node(ChHTNode* node);

        //ChHTNode* find_node(const T& Data) const;
        ChHTNode* find_node(const unsigned& i, const char* Key, ChHTNode*& prev_node) const;
        void insert_node(const unsigned& i, ChHTNode* node);
};
```

```
135
136 #include "ChHashTable.cpp"
137
138 #endif
```