```
1 /*****************************************************************************
2  File name:  hashtable.h
3  Author:     $Author: chadd $
4  Date:       Oct 21, 2011
5  Class:
6  Assignment:
7  Purpose:
8  RevisionID:  $Id: hashtable.h 173 2011-11-14 23:59:22Z chadd $
9  *****************************************************************************/
10
11
12 #ifndef HASHTABLE_H_
13 #define HASHTABLE_H_
14
15 #include "../../CS300DynamicList/include/list.h"
16
17 /*typedef struct HT_DATATYPE
18 {
19   int count;
20 } HT_DATATYPE;
21
22 In DATATYPE:
23
24   char ht_key[100];
25   unsigned int ht_size;
26   HT_DATATYPE ht_data;
27
28 */
29
30 #define HT_NO_ERROR                   0
31
32 // list create failed
33 #define HT_ERROR_NO_HASHTABLE_CREATE  -1
34
35 // user tried to add data to a full hash table
36 #define HT_ERROR_FULL_HASHTABLE       -3
37
38 // user tried to operate on an invalid hash table. An invalid
39 // hash table may be a NULL psTable
40 #define HT_ERROR_INVALID_HASHTABLE    -9
41
42 // user provided a NULL pointer to the function (other than the hash table)
43 #define HT_ERROR_NULL_PTR             -10
44
45 // userHashFunction failed
46 #define HT_ERROR_HASH_FAILED          -11
47
48 // the key did not exists
49 #define HT_ERROR_NO_DATA              -12
50
51 // the key already exists
52 #define HT_ERROR_DUPLICATE            -13
53
54 typedef struct HashTable
55 {
56   unsigned int tableSize;
57   List *psTable;
58
59   /*
```

```
60    * userHashFunction
61    *
62    * param:
63    *  char*  the data over which to calculate the hash
64    *  unsigned int  the size of the data
65    *  unsigned int* the hash value produced
66    *  unsigned int  the size of the hash table (for % size calculations)
67    *
68    *  MUST produce a value between 0 and tableSize-1 INCLUSIVE
69    *
70    * return:
71    *  TRUE success
72    *  FALSE failure of any kind
73    */
74   int (*userHashFunction) (char*, unsigned int, unsigned int*, unsigned int);
75
76   /*
77    * userCompare:
78    *  return < 0 if left <  right
79    *  return   0 if left == right
80    *  return > 0 if left >  right
81    *
82    *  Longer strings are greater
83    *
84    *  char*  the left key
85    *  unsigned int  the size of the left key data
86    *  char*  the right key
87    *  unsigned int  the size of the right key data
88    */
89   int (*userCompare) (char*, unsigned int, char*, unsigned int);
90 } HashTable;
91
92 typedef HashTable* HashTablePtr;
93
94 typedef struct HT_Stats
95 {
96   unsigned int maxChain;
97   unsigned int numberElements;
98   float averageAccessTime;
99   unsigned int emptyBuckets;
100 } HT_Stats;
101
102 /*****************************************************************************
103 *                    Allocation and Deallocation
104 *****************************************************************************/
105 ERRORCODE htCreate(HashTablePtr* hHashTable, unsigned int size,
106     int (*userHashFunction) (char*, unsigned int, unsigned int*, unsigned int),
107     int (*userCompare) (char*, unsigned int, char*, unsigned int)
108     );
109 // HT_ERROR_NO_HASHTABLE_CREATE
110
111 ERRORCODE htDispose(HashTablePtr* hHashTable);
112 // HT_ERROR_INVALID_HASHTABLE
113
114 /*****************************************************************************
115 *                  Checking for elements
116 *****************************************************************************/
117 ERRORCODE htContains(HashTablePtr psHashTable, char*, unsigned int size,
118     HT_DATATYPE *result);
```

```
119 // HT_ERROR_INVALID_HASHTABLE
120 // HT_ERROR_NULL_PTR
121 // HT_ERROR_HASH_FAILED
122 // HT_ERROR_NO_DATA
123
124 ERRORCODE htIsEmpty(HashTable sHashTable, BOOLEAN *);
125 // HT_ERROR_INVALID_HASHTABLE
126 // HT_ERROR_NULL_PTR
127
128 ERRORCODE htGetStats(HashTable sHashTable, HT_Stats*);
129 // HT_ERROR_INVALID_HASHTABLE
130 // HT_ERROR_NULL_PTR
131
132 /*****************************************************************************
133 *                 Insertion, Deletion, and Updating
134 *****************************************************************************/
135 ERRORCODE htInsert(HashTablePtr psHashTable, char*, unsigned int size,
136     HT_DATATYPE data);
137 // HT_ERROR_INVALID_HASHTABLE
138 // HT_ERROR_NULL_PTR
139 // HT_ERROR_HASH_FAILED
140 // HT_ERROR_DUPLICATE
141 // HT_ERROR_FULL_HASHTABLE
142
143 ERRORCODE htDelete(HashTablePtr psHashTable, char*, unsigned int size,
144     HT_DATATYPE *data);
145 // HT_ERROR_INVALID_HASHTABLE
146 // HT_ERROR_NULL_PTR
147 // HT_ERROR_HASH_FAILED
148 // HT_ERROR_NO_DATA
149
150 ERRORCODE htUpdate(HashTablePtr psHashTable, char *, unsigned int size,
151     HT_DATATYPE result);
152 // HT_ERROR_INVALID_HASHTABLE
153 // HT_ERROR_NULL_PTR
154 // HT_ERROR_HASH_FAILED
155 // HT_ERROR_NO_DATA
156
157 /*****************************************************************************
158 *                 Visitor Functions
159 *****************************************************************************/
160
161 ERRORCODE htVisit(HashTablePtr psHashTable,
162     void (*visitor)(char*, unsigned int, HT_DATATYPE));
163 /*
164  * the parameters for visitor are:
165  * char* the key
166  * unsigned int the length of the key
167  * HT_DATATYPE the entry in the hash table
168  */
169 // HT_ERROR_INVALID_HASHTABLE
170 // HT_ERROR_NULL_PTR
171
172 ERRORCODE htVisitAndUpdate(HashTablePtr psHashTable,
173     void (*visitor)(char*, unsigned int, HT_DATATYPE*));
174 /*
175  * the parameters for visitor are:
176  * char* the key
177  * unsigned int the length of the key
```

```
178  * HT_DATATYPE* the updatable entry in the hash table
179  */
180 // HT_ERROR_INVALID_HASHTABLE
181 // HT_ERROR_NULL_PTR
182
183
184 #endif /* HASHTABLE_H_ */
185
```