```
1   // Arup Guha
2   // 3/31/08
3   // Hash Table of words implemented using Linear Probing.
4
5   #include <stdio.h>
6   #include <string.h>
7   #include <math.h>
8
9   // Note: This constrains the limits of static memory allocation...
10  #define MAX_SIZE 29
11  #define TABLE_SIZE 59999
12
13  struct htable {
14      char entries[TABLE_SIZE][MAX_SIZE+1];
15  };
16
17  // Hash Table Functions.
18  void initTable(struct htable *h);
19  int hashvalue(char word[]);
20  void insertTable(struct htable *h, char word[]);
21  int searchTable(struct htable *h, char word[]);
22  void deleteTable(struct htable *h, char word[]);
23
24  int main() {
25
26      char filename[MAX_SIZE+1], temp[MAX_SIZE+1];
27      FILE *ifp;
28      int numwords, i;
29      struct htable mytable;
30      int ans;
31
32      // Get the file name.
33      printf("What is the name of the dictionary file?\n");
34      scanf("%s", &filename);
35      ifp = fopen(filename, "r");
36
37      fscanf(ifp, "%d", &numwords);
38
39      // Read in all of the words from a file into the table.
40      // This is only done once.
41      printf("get here\n");
42      initTable(&mytable);
43      printf("init table\n");
44
45      for (i=0; i<numwords; i++) {
46          fscanf(ifp, "%s", temp);
47          insertTable(&mytable, temp);
48      }
49
50
51      // Allow the user to make changes to the hash table and search for words.
52      do {
53
54          printf("Do you want to 1)search word, 2) add word, 3) delete a word?\n");
55          scanf("%d", &ans);
56
57          // Search for a word.
58          if (ans == 1) {
59
60              printf("What word are you looking for?\n");
61              scanf("%s", temp);
62              if (searchTable(&mytable, temp))
63                  printf("%s was found.\n", temp);
64              else
65                  printf("%s was NOT found.\n", temp);
66          }
67
68          // Add a word.
69          else if (ans == 2) {
70
71              printf("What word do you want to add?\n");
72              scanf("%s", temp);
73              if (searchTable(&mytable, temp))
74                  printf("%s was ALREADY in the table\n", temp);
75              else
76                  insertTable(&mytable, temp);
77          }
78
79          // Delete a word.
80          else if (ans == 3) {
81
82              printf("What word do you want to delete?\n");
83              scanf("%s", temp);
84              deleteTable(&mytable, temp);
```

```
85
86          }
87
88       } while (ans < 4); // Not very user friendly, just quits for any number > 3.
89
90       system("PAUSE");
91       return 0;
92    }
93
94    // Pre-condition: none
95    // Post-condition: Sets each entry in the hash table pointed to by h to the
96    //                 empty string.
97    void initTable(struct  htable  *h) {
98       int  i;
99
100      // Our marker for an empty entry is the empty string.
101      for (i=0; i<TABLE_SIZE; i++)
102          strcpy(h->entries[i], "");
103   }
104
105   // Pre-condition: none
106   // Post-condition: Calculates a hash value for word.
107   int  hashvalue(char  word[]) {
108
109      int  i, sum=0;
110
111      // Basically represents the value of word in base 128 (according to ascii
112      // values) and returns its value mod the TABLE_SIZE.
113      for (i=0; i<strlen(word); i++)
114         sum = (128*sum + (int)(word[i]))%TABLE_SIZE;
115
116      return sum;
117   }
118
119   // Pre-condition: h points to a valid hash table that IS NOT full.
120   // Post-condition: word will be inserted into the table h.
121   void insertTable(struct  htable  *h, char  word[]) {
122
123      int  hashval, quad = 1;
124      hashval = hashvalue(word);
125
126      // Here's the linear probing part.
127      while (strcmp(h->entries[hashval], "") != 0)
128      {
129                hashval = (hashval+(quad*quad))%TABLE_SIZE;
130                quad++;
131      }
132
133      strcpy(h->entries[hashval], word);
134   }
135
136   // Pre-condition: h points to a valid hash table.
137   // Post-condition: 1 will be returned iff word is stored in the table pointed to
138   //                 by h. Otherwise, 0 is returned.
139   int  searchTable(struct  htable  * h, char  word[]) {
140
141      int  hashval, quad = 1;
142      hashval = hashvalue(word);
143
144      // See what comes first, the word or a blank spot.
145      while (strcmp(h->entries[hashval], "") != 0 &&
146            strcmp(h->entries[hashval], word) != 0)
147            {
148                        hashval = (hashval+(quad*quad))%TABLE_SIZE;
149                         quad++;
150            }
151
152      // The word was in the table.
153      if (strcmp(h->entries[hashval], word) == 0)
154         return 1;
155
156      // It wasn't.
157      return 0;
158
159   }
160
161   // Pre-condition: h points to a valid hash table.
162   // Post-condition: deletes word from the table pointed to by h, if word is
163   //                 stored here. If not, no change is made to the table pointed
164   //                 to by h.
165   void deleteTable(struct  htable  *h, char  word[]) {
166
167      int  hashval, quad = 1;
168      hashval = hashvalue(word);
```

```
169
170       // See what comes first, the word or a blank spot.
171       while (strcmp(h->entries[hashval], "") != 0 &&
172              strcmp(h->entries[hashval], word) != 0)
173              {
174                          hashval = (hashval+(quad*quad))%TABLE_SIZE;
175                           quad++;
176              }
177
178       // Reset the word to be the empty string.
179       if (strcmp(h->entries[hashval], word) == 0)
180           strcpy(h->entries[hashval],"");
181
182       // If we get here, the word wasn't in the table, so nothing is done.
183   }
184
```