```
// Trever Wagenhals
// Program 10

#ifndef HASH_TABLE
#define HASH_TABLE

#include <iostream>
#include <iomanip>

using namespace std;

#include "CursorCntl.h"
#include "LinkedList_T.h"

template <typename ItemData, unsigned NumBuckets>
class HashTable
{
public:
   // Constructor: Make all buckets empty.
   HashTable() : size(0), curBucket(0) {}

   // Search the table for entry "d." If "d" is found, return true,make "d"
   // the current entry for this bucket, and make this the current bucket.
   // If "d" is not found, return false.
   bool Search(ItemData &d);
   // Add an entry to the table.
   void Insert(ItemData &d);
   // Update an entry in the table.
   void Update(ItemData &d)
   {
      unsigned index = Hash(d); // Compute bucket index.
      bucket[index].Update(d);  // Update the current entry in this bucket.
   }

   void Delete();    // Delete the current entry.
   // Return true if the table is empty.
   bool Empty() { return size == 0; }
   // Return the number of entries in the table.
   unsigned Size() { return size; }
   // Return the current entry.
   ItemData CurrentEntry() { return bucket[curBucket].CurrentEntry(); }
   // Remove the first entry from the table.
   ItemData Remove();
   // Show the hash table on the right side of the screen.
   void Show();
private:
   unsigned size;       // The number of entries in the table.
   unsigned curBucket;  // The index of the current bucket
   LinkedList<ItemData> bucket[NumBuckets]; // Array of pointers to linked lists
   unsigned long Hash(ItemData &d);         // The hashing function
};

// Obtain a hash key from the ItemData object "d" and then generate
// a random table index such that 0 <= index <= Num Buckets-1.
template <typename ItemData, unsigned NumBuckets>
unsigned long HashTable<ItemData, NumBuckets>::Hash(ItemData &d)
{
   unsigned x = d.HashKey();    // Ask ItemData "d" for its hash key.

   // Now, generate a random table index such that 0 <= index <= NumBuckets-1
   const unsigned C1 = 25173;
   const unsigned C2 = 13849;
   const unsigned C3 = 65536;
   return ((C1*x + C2) % C3) % NumBuckets;
}
```

```
// Display a the hash table on the right half of the screen.
template <typename ItemData, unsigned NumBuckets>
void HashTable<ItemData, NumBuckets>::Show()
{
#if !NoGraphics
    const unsigned XLeft = 40;          // Column number for start of dictionary display
    const unsigned XHeading = XLeft - 3;// Column location of heading
    const unsigned ScrollsAt = 24;      // Screen scrolls after line 24
    const unsigned XMax = 79;           // Don't show words after this column
    const unsigned YSpacing = NumBuckets < 22 ? 22 / NumBuckets : 1;  // Vertical spacing
    const unsigned DisplayLines = NumBuckets < 22 ? NumBuckets : 22;// Number of buckets to display

    int xOld;                       // Old cursor position x coordinate
    int yOld;                       // Old cursor position y coordinate

                                    // Save cursor position
    getxy(xOld, yOld);

    // Has the screen scrolled yet?
    int deltaY = 0;

    if (yOld > ScrollsAt)
        deltaY = yOld - ScrollsAt + 1;

    // Clear the right half of the screen.
    for (int y = 0; y<ScrollsAt + 1; y++)
    {
        gotoxy(XLeft, y + deltaY);
        clreol();
    }

    // Display heading.
    gotoxy(XHeading, deltaY);
    cout << "BUCKET";

    // Show the array and offset if scrolled.
    for (unsigned index = 0; index<DisplayLines; index++)
    {
        // Display the bucket number.
        gotoxy(XLeft, YSpacing*index + deltaY + 2);
        cout << setw(2) << right << index << ": ";

        // Traverse the linked list bucket,
        // displaying each entry.
        bucket[index].Rewind();
        while (!bucket[index].AtEnd())
        {
            int xCursor;  // cursor x position
            int yCursor;  // cursor y position

            // Don't go off the right side of the screen
            getxy(xCursor, yCursor);
            if (xCursor + bucket[index].CurrentEntry().Word().length() >= XMax)
                break;

            // Display the next entry from the bucket.
            cout << left;
            bucket[index].CurrentEntry().Show();
         cout << " ";
            bucket[index].Skip();
        }
    }
    gotoxy(xOld, yOld);     // Restore old cursor position.
#endif
}
```

```
131
132 // Remove the first word from the first bucket with data
133 template <typename ItemData, unsigned NumBuckets>
134 ItemData HashTable<ItemData, NumBuckets>::Remove()
135 {
136     // Loop through each bucket
137     for (curBucket = 0; curBucket < NumBuckets; curBucket++)
138     {
139         if (!bucket[curBucket].Empty())     // If bucket isn't empty
140         {
141             bucket[curBucket].Rewind();     // Reset bucket
142             ItemData temp = CurrentEntry();
143             bucket[curBucket].Delete();     // Delete 1st bucket entry
144             size--;
145             return temp;
146         }
147     }
148     return CurrentEntry(); // Should never occur, but removes warning messages
149 }
150
151 // Determine if the data's hashed bucket contains the data already
152 template <typename ItemData, unsigned NumBuckets>
153 bool HashTable<ItemData, NumBuckets>::Search(ItemData &d)
154 {
155     curBucket = Hash(d);
156     if (!bucket[curBucket].Empty()) // Skip if bucket empty
157     {
158         bucket[curBucket].Rewind();         // Rewind to check all entries
159         while (!bucket[curBucket].AtEnd())  // Check all entries
160         {
161             if (bucket[curBucket].CurrentEntry() == d)
162                 return true;
163             else
164                 bucket[curBucket].Skip();
165         }
166     }
167     return false;
168 }
169
170 // Insert data into data's hashed bucket and increment size
171 template <typename ItemData, unsigned NumBuckets>
172 void HashTable<ItemData, NumBuckets>::Insert(ItemData &d)
173 {
174     bucket[Hash(d)].Insert(d);
175     size++;
176 }
177
178 // Remove current data from current bucket and decrement size
179 template <typename ItemData, unsigned NumBuckets>
180 void HashTable<ItemData, NumBuckets>::Delete()
181 {
182     bucket[curBucket].Delete();
183     size--;
184 }
185
186 #endif
```