```
1  //  Created by Frank M. Carrano and Tim Henry.
2  //  Copyright (c) 2013 __Pearson Education__. All rights reserved.
3
4  /** ADT bag: Link-based implementation.
5      @file LinkedBag.cpp */
6
7  #include "LinkedBag.h"
8  #include "Node.h"
9  #include <cstddef>
10
11 template<class ItemType>
12 LinkedBag<ItemType>::LinkedBag()
13 {
14     itemCount = 0;
15     head_ptr = nullptr;
16 }  // end default constructor
17
18 template<class ItemType>
19 LinkedBag<ItemType>::LinkedBag(const LinkedBag<ItemType>& aBag)
20 {
21     itemCount = aBag.itemCount;
22    Node<ItemType>* origChainPtr = aBag.headPtr;  // Points to nodes in origina
   l chain
23
24    if (origChainPtr == nullptr)
25       head_ptr = nullptr;  // Original bag is empty
26    else
27    {
28       // Copy first node
29       head_ptr = new Node<ItemType>();
30       head_ptr->setItem(origChainPtr->getItem());
31
32       // Copy remaining nodes
33       Node<ItemType>* newChainPtr = head_ptr;      // Points to last node in n
   ew chain
34       origChainPtr = origChainPtr->getNext();     // Advance original-chain po
   inter
35
36       while (origChainPtr != nullptr)
37       {
38          // Get next item from original chain
39          ItemType nextItem = origChainPtr->getItem();
40
41          // Create a new node containing the next item
42          Node<ItemType>* newNodePtr = new Node<ItemType>(nextItem);
43
44          // Link new node to end of new chain
45          newChainPtr->setNext(newNodePtr);
46
47          // Advance pointer to new last node
48          newChainPtr = newChainPtr->getNext();
49
50          // Advance original-chain pointer
51          origChainPtr = origChainPtr->getNext();
52       }  // end while
53
54       newChainPtr->setNext(nullptr);              // Flag end of chain
55    }  // end if
```

```
 56 }  // end copy constructor
 57
 58 template<class ItemType>
 59 LinkedBag<ItemType>::~LinkedBag()
 60 {
 61    clear();
 62 }  // end destructor
 63
 64 template<class ItemType>
 65 bool LinkedBag<ItemType>::isEmpty() const
 66 {
 67     return (head_ptr == nullptr);
 68 }  // end isEmpty
 69
 70 template<class ItemType>
 71 bool LinkedBag<ItemType>::contains(const ItemType& anItem) const
 72 {
 73     Node<ItemType>* curr = head_ptr;
 74     while (curr != nullptr)
 75     {
 76         if (curr->getItem() == anItem)
 77             return true;
 78         else
 79             curr = curr->getNext();
 80     }
 81     return false;
 82 }  // end isEmpty
 83
 84 template<class ItemType>
 85 bool LinkedBag<ItemType>::isFull() const
 86 {
 87     return false;
 88 }  // end isEmpty
 89
 90 template<class ItemType>
 91 int LinkedBag<ItemType>::getCapacity() const
 92 {
 93     return getCurrentSize();
 94 }  // end isEmpty
 95
 96
 97 template<class ItemType>
 98 int LinkedBag<ItemType>::getCurrentSize() const
 99 {
100     return itemCount;
101 }  // end getCurrentSize
102
103 template<class ItemType>
104 bool LinkedBag<ItemType>::add(const ItemType& newEntry)
105 {
106    // Add to beginning of chain: new node references rest of chain;
107    // (headPtr is null if chain is empty)
108     if (head_ptr == nullptr)
109         head_ptr = new Node<ItemType>(newEntry);
110     else
111         head_ptr = new Node<ItemType>(newEntry, head_ptr);
112     ++itemCount;
113     return true;
```

```
114 }  // end add
115
116 template<class ItemType>
117 bool LinkedBag<ItemType>::remove(const ItemType& anEntry)
118 {
119     Node<ItemType>* to_delete;
120
121     if (head_ptr == nullptr) return false;
122
123     if (head_ptr->getItem() == anEntry)
124     {
125         to_delete = head_ptr;
126         head_ptr = head_ptr->getNext();
127         delete to_delete;
128         --itemCount;
129         return true;
130     }
131
132     Node<ItemType>* curr = head_ptr;
133     while (curr != nullptr && curr->getNext() != nullptr)
134     {
135         if (curr->getNext()->getItem() == anEntry)
136         {
137             to_delete = curr->getNext();
138             curr->setNext(to_delete->getNext());
139             delete to_delete;
140             --itemCount;
141             return true;
142         }
143         curr = curr->getNext();
144     }
145     return false;
146 }  // end remove
147
148 template<class ItemType>
149 void LinkedBag<ItemType>::clear()
150 {
151    Node<ItemType>* nodeToDeletePtr = head_ptr;
152    while (head_ptr != nullptr)
153    {
154       head_ptr = head_ptr->getNext();
155
156       // Return node to the system
157       nodeToDeletePtr->setNext(nullptr);
158       delete nodeToDeletePtr;
159
160       nodeToDeletePtr = head_ptr;
161    }  // end while
162    // headPtr is nullptr; nodeToDeletePtr is nullptr
163
164     itemCount = 0;
165 }  // end clear
166
167
168 template<class ItemType>
169 int LinkedBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
170 {
171     int frequency = 0;
```

```
172     int counter = 0;
173     Node<ItemType>* curPtr = head_ptr;
174     while ((curPtr != nullptr) && (counter < itemCount))
175     {
176        if (anEntry == curPtr->getItem())
177        {
178           frequency++;
179        } // end if
180
181        counter++;
182        curPtr = curPtr->getNext();
183     } // end while
184
185      return frequency;
186  }  // end getFrequencyOf
187
188  // private
189  // Returns either a pointer to the node containing a given entry
190  // or the null pointer if the entry is not in the bag.
191  template<class ItemType>
192  Node<ItemType>* LinkedBag<ItemType>::getPointerTo(const ItemType& anEntry) con
     st
193  {
194     bool found = false;
195     Node<ItemType>* curPtr = head_ptr;
196
197     while (!found && (curPtr != nullptr))
198     {
199        if (anEntry == curPtr->getItem())
200           found = true;
201        else
202           curPtr = curPtr->getNext();
203     } // end while
204
205     return curPtr;
206  } // end getPointerTo
207
208
209  template<class ItemType>
210  vector<ItemType> LinkedBag<ItemType>::toVector() const
211  {
212      vector<ItemType> stuff;
213
214      Node<ItemType>* curr = head_ptr;
215      while (curr != nullptr)
216      {
217          stuff.push_back(curr->getItem());
218          curr = curr->getNext();
219      }
220      return stuff;
221  }  // end isEmpty
222
```