```
 1  /*****************************************************************************
 2   * AUTHOR           : Nick Reardon
 3   * Assignment #4    : Deque To Queue
 4   * CLASS            : CS1D
 5   * SECTION          : MW - 2:30p
 6   * DUE DATE         : 02 / 10 / 20
 7   *****************************************************************************/
 8  #ifndef _ARRAYHEAP_H_
 9  #define _ARRAYHEAP_H_
10  #include <exception>
11  #include "Except.h"
12
13  enum ERROR_TYPE
14  {
15      DEFUALT,
16      FULL,
17      EMPTY,
18      OUT_OF_RANGE
19  };
20
21  template <typename Type, typename Key>
22  struct heapMember
23  {
24      Type value;
25      Key key;
26
27      heapMember(const Type& newValue, const Key& newKey)
28      {
29          value = newValue;
30          key = newKey;
31      }
32
33      heapMember() {}
34
35      void swap(heapMember& other)
36      {
37          Key tempKey;
38          Type tempType;
39
40          tempKey = other.key;
41          tempType = other.value;
42
43          other.key = this->key;
44          other.value = this->value;
45
46          this->key = tempKey;
47          this->value = tempType;
48
49
50      }
51
52      inline bool operator< (const heapMember& other)
```

```
53      {
54          return (this->key < other.key);
55      }
56
57      inline bool operator> (const heapMember& other)
58      {
59          return (this->key > other.key);
60      }
61
62      inline void operator= (const heapMember& other)
63      {
64          this->key = other.key;
65          this->value = other.value;
66      }
67  };
68
69  template <class Type, class Key>
70  class ArrayMaxHeap
71  {
72  private:
73      heapMember<Type, Key>* heap;
74
75      int currentSize;
76
77      int capacity;
78
79  protected:
80      void sort()
81      {
82          int index = 1;
83
84          int swapIndex;
85
86          while ( (2 * index < currentSize) &&
87              ((heap[index].key < heap[2 * index].key) || (heap[index].key < heap[2
                * index + 1].key)) )
88          {
89              if (heap[2 * index].key > heap[2 * index + 1].key)
90              {
91                  swapIndex = 2 * index;
92              }
93              else
94              {
95                  swapIndex = 2 * index + 1;
96              }
97
98              heap[index].swap(heap[swapIndex]);
99
100             index = swapIndex;
101         }
102     }
103
```

```
104
105
106  public:
107
108      ArrayMaxHeap<Type,Key>(const int newCapacity = 32)
109      {
110          heap = new heapMember<Type, Key>[newCapacity];
111          currentSize = 0;
112          capacity = newCapacity;
113      }
114
115      //VectorHeap<Type>(const VectorHeap<Type>& otherDeque);
116
117      ~ArrayMaxHeap()
118      {
119          delete[] heap;
120      }
121
122      bool empty() const { return currentSize == 0; }
123
124      bool full() const { return currentSize == capacity; }
125
126      int size() const { return size; }
127
128      void insert(const Type& element, const Key& newKey)
129      {
130          if (full())
131          {
132              throw Except("container is full", FULL, 5);
133          }
134
135          currentSize++;
136
137          heap[currentSize].value = element;
138          heap[currentSize].key = newKey;
139
140
141          int index = currentSize;
142
143          while ( (heap[index].key > heap[index / 2].key) && ((index / 2) != 0) )
144          {
145              heap[index].swap(heap[index / 2]);
146
147              index /= 2;
148
149          }
150
151      }
152
153      void remove()
154      {
155          if (empty())
```

```
156         {
157             throw(Except("Cannot remove - heap is empty", EMPTY, 5));
158         }
159 
160         heap[1] = heap[currentSize];
161         currentSize--;
162         sort();
163     }
164 
165 
166 
167     Type max() const
168     {
169         Type temp = heap[1].value;
170         return temp;
171     }
172 
173     void printAll(std::ostream& output) const
174     {
175         if (empty())
176         {
177             throw(Except("Cannot print - heap is empty", EMPTY, 5));
178         }
179 
180         int current = 0;
181         int levelSize = 1;
182 
183         for (int i = 1; i < currentSize; i++)
184         {
185             output << heap[i].value << '(' << heap[i].key << ')' << "  ";
186 
187             current++;
188 
189             if (current == levelSize)
190             {
191                 current = 0;
192 
193                 levelSize = levelSize * 2;
194 
195                 output << '\n';
196             }
197         }
198 
199         output << "\n\n";
200     }
201 
202 };
203 
204 
205 #endif //!_ARRAYHEAP_H_
```