```
/* SkewHeap.c */
1
 2
     #include <stdlib.h>
 3
     #include <malloc.h>
4
     #include <string.h>
5
     #include <assert.h>
     #include "SkewHeap.h"
6
 7
     #include "LinkQueue.h"
9
     static BINNODE *nodeNew(int keySize, const void *e)
10
         BINNODE *newNode = (BINNODE *) malloc(sizeof(BINNODE) + keySize);
11
12
         if (NULL == newNode)
13
         {
14
             return NULL;
15
         }
16
         newNode->parent = NULL;
17
         newNode->lc = NULL;
18
         newNode->rc = NULL;
19
         memcpy(newNode->key, e, keySize);
20
         return newNode;
21
     }
22
23
     static void nodeDispose (BINNODE *node, SkewHeapFree *freeFn)
24
25
         if (NULL != freeFn)
26
         {
27
             freeFn (node->key);
28
29
         free (node);
30
     }
31
32
     //PQueue初始化
33
     void PQueueNew (PQUEUE *pq, int keySize, SkewHeapCmp *cmpFn, SkewHeapFree *freeFn)
34
35
         assert(0 < keySize);</pre>
36
         assert (NULL != cmpFn);
37
         pq->keySize = keySize;
38
         pq->size = 0;
39
         pq->cmpFn = cmpFn;
40
         pq->freeFn = freeFn;
41
         pq->root = NULL;
42
     }
43
44
     //PQueue判空
45
     int PQueueEmpty(PQUEUE *pq)
46
     {
47
         return (0 == pq->size);
48
     }
49
     //PQueue规模
50
51
     int PQueueSize(PQUEUE *pq)
52
     {
53
         return pq->size;
54
     }
55
     static void addNode2Queue(void *elemAddr, void *outData)
56
57
58
         QUEUE *q = (QUEUE *) outData;
59
         if (NULL == q)
60
         {
61
             return ;
62
63
         QueueEn(q, elemAddr);
64
     }
65
66
     //PQueue销毁
67
     void PQueueDispose(PQUEUE *pq)
68
69
         if (PQueueEmpty(pq))
70
         {
71
             return ;
73
         QUEUE nodeQueue;
```

```
74
          QueueNew(&nodeQueue, sizeof(BINNODE *), NULL);
 75
          BINNODE *node = pq->root;
 76
          QueueEn(&nodeQueue, &node);
 77
          while (!QueueEmpty(&nodeQueue))
 78
 79
              QueueDe (&nodeQueue, &node);
 80
              if (NULL != node->lc)
 81
              {
 82
                   QueueEn(&nodeQueue, &(node->lc));
 83
              1
 84
              if (NULL != node->rc)
 85
              -{
                   QueueEn(&nodeQueue, &(node->rc));
 87
              }
 88
              nodeDispose(node, pq->freeFn);
 89
 90
          QueueDispose (&nodeQueue);
 91
          pq->root = NULL;
 92
          pq->size = 0;
 93
      }
 94
 95
      //获取当前优先级最大的元素
      int PQueueGetMax(PQUEUE *pq, void *e)
 96
 97
      {
 98
          if (PQueueEmpty(pq) || NULL == e)
 99
          {
100
              return -1;
101
          }
102
          memcpy(e, pq->root->key, pq->keySize);
103
          return 0;
104
      }
105
      //合并以a和b为根节点的两个斜堆
106
107
      static BINNODE *merge(PQUEUE *pq, BINNODE *a, BINNODE *b)
108
109
          if (NULL == a)
110
          {
111
              return b;
112
          1
113
          if (NULL == b)
114
          {
115
              return a:
116
          }
117
          if (0 > pq - cmpFn(a - key, b - key))
118
          {
119
              BINNODE *tmp = a;
120
              a = b;
121
              b = tmp;
122
          }
123
          a \rightarrow rc = merge(pq, a \rightarrow rc, b);
124
          a \rightarrow rc \rightarrow parent = a;
125
          //交换以a为根的左右分支
          BINNODE *tmp = a->lc;
126
127
          a->lc= a->rc;
          a \rightarrow rc = tmp;
128
129
          return a;
130
      }
131
132
      //优先级队列插入关键码e,返回值:0--成功,!0--失败
133
      int PQueueInsert(PQUEUE *pq, const void *e)
134
135
          BINNODE *newNode = nodeNew(pq->keySize, e);
136
          if (NULL == newNode)
137
          {
138
              return -1;
139
          }
140
          pq->root = merge(pq, pq->root, newNode);
141
          pq->root->parent = NULL;
142
          pq->size ++;
143
          return 0;
144
      }
145
146
      //优先级队列删除优先级最大的元素
```

```
int PQueueDeleteMax(PQUEUE *pq)
147
148
149
          if (PQueueEmpty(pq))
150
          {
151
              return -1;
152
          }
153
          BINNODE *lHeap = pq->root->lc;
154
          BINNODE *rHeap = pq->root->rc;
155
          nodeDispose(pq->root, pq->freeFn);
          pq->size --;
pq->root = merge(pq, lHeap, rHeap);
156
157
158
          if (NULL != pq->root)
159
          {
160
              pq->root->parent = NULL;
161
          }
162
          return 0;
163
     }
```