```c
/* LeftHeap.c */
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <assert.h>
#include "LeftHeap.h"
#include "LinkQueue.h"

static BINNODE *nodeNew(int keySize, const void *e)
{
    BINNODE *newNode = (BINNODE *)malloc(sizeof(BINNODE) + keySize);
    if (NULL == newNode)
    {
        return NULL;
    }
    newNode->parent = NULL;
    newNode->lc = NULL;
    newNode->rc = NULL;
    newNode->npl = 0;
    memcpy(newNode->key, e, keySize);
    return newNode;
}

static void nodeDispose(BINNODE *node, LeftHeapFree *freeFn)
{
    if (NULL != freeFn)
    {
        freeFn(node->key);
    }
    free(node);
}

//PQueue初始化
void PQueueNew(PQUEUE *pq, int keySize, LeftHeapCmp *cmpFn, LeftHeapFree *freeFn)
{
    assert(0 < keySize);
    assert(NULL != cmpFn);
    pq->keySize = keySize;
    pq->size = 0;
    pq->cmpFn = cmpFn;
    pq->freeFn = freeFn;
    pq->root = NULL;
}

//PQueue判空
int PQueueEmpty(PQUEUE *pq)
{
    return (0 == pq->size);
}

//PQueue规模
int PQueueSize(PQUEUE *pq)
{
    return pq->size;
}

static void addNode2Queue(void *elemAddr, void *outData)
{
    QUEUE *q = (QUEUE *)outData;
    if (NULL == q)
    {
        return ;
    }
    QueueEn(q, elemAddr);
}

//PQueue销毁
void PQueueDispose(PQUEUE *pq)
{
    if (PQueueEmpty(pq))
    {
        return ;
    }
```

```
74      QUEUE nodeQueue;
75      QueueNew(&nodeQueue, sizeof(BINNODE *), NULL);
76      BINNODE *node = pq->root;
77      QueueEn(&nodeQueue, &node);
78      while (!QueueEmpty(&nodeQueue))
79      {
80          QueueDe(&nodeQueue, &node);
81          if (NULL != node->lc)
82          {
83              QueueEn(&nodeQueue, &(node->lc));
84          }
85          if (NULL != node->rc)
86          {
87              QueueEn(&nodeQueue, &(node->rc));
88          }
89          nodeDispose(node, pq->freeFn);
90      }
91      QueueDispose(&nodeQueue);
92      pq->root = NULL;
93      pq->size = 0;
94  }
95
96  //获取当前优先级最大的元素
97  int PQueueGetMax(PQUEUE *pq, void *e)
98  {
99      if (PQueueEmpty(pq) || NULL == e)
100     {
101         return -1;
102     }
103     memcpy(e, pq->root->key, pq->keySize);
104     return 0;
105 }
106
107 //合并以a和b为根节点的两个左式堆
108 static BINNODE *merge(PQUEUE *pq, BINNODE *a, BINNODE *b)
109 {
110     if (NULL == a)
111     {
112         return b;
113     }
114     if (NULL == b)
115     {
116         return a;
117     }
118     if (0 > pq->cmpFn(a->key, b->key))
119     {
120         BINNODE *tmp = a;
121         a = b;
122         b = tmp;
123     }
124     a->rc = merge(pq, a->rc, b);
125     a->rc->parent = a;
126     if (NULL == a->lc || a->lc->npl < a->rc->npl)
127     {
128         BINNODE *tmp = a->lc;
129         a->lc= a->rc;
130         a->rc = tmp;
131     }
132     a->npl = (NULL == a->rc) ? 1 : a->rc->npl + 1;
133     return a;
134 }
135
136 //优先级队列插入关键码e，返回值：0--成功，!0--失败
137 int PQueueInsert(PQUEUE *pq, const void *e)
138 {
139     BINNODE *newNode = nodeNew(pq->keySize, e);
140     if (NULL == newNode)
141     {
142         return -1;
143     }
144     pq->root = merge(pq, pq->root, newNode);
145     pq->root->parent = NULL;
146     pq->size ++;
```

```
147         return 0;
148     }
149
150     //优先级队列删除优先级最大的元素
151     int PQueueDeleteMax(PQUEUE *pq)
152     {
153         if (PQueueEmpty(pq))
154         {
155             return -1;
156         }
157         BINNODE *lHeap = pq->root->lc;
158         BINNODE *rHeap = pq->root->rc;
159         nodeDispose(pq->root, pq->freeFn);
160         pq->size --;
161         pq->root = merge(pq, lHeap, rHeap);
162         if (NULL != pq->root)
163         {
164             pq->root->parent = NULL;
165         }
166         return 0;
167     }
```