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

//判断PQ[i]是否合法
#define InHeap(n, i)        (((-1) < (i)) && ((i) < (n)))
//PQ[i]的父亲节点
#define Parent(i)           ((i - 1) >> 1)
//最后一个内部节点（末节点的父亲）
#define LastInternal(n)     Parent(n - 1)
//PQ[i]的左孩子
#define LChild(i)           (1 + ((i) << 1))
//PQ[i]的右孩子
#define RChild(i)           ((1 + (i)) << 1)
//判断PQ[i]是否有父亲
#define ParentValid(i)      (0 < i)
//判断PQ[i]是否有一个（左）孩子
#define LChildVaild(n, i)   InHeap(n, LChild(i))
//判断PQ[i]是否有两个孩子
#define RChildVaild(n, i)   InHeap(n, RChild(i))

//优先级队列堆空间倍增
static void PQueueGrow(PQUEUE *pq)
{
    pq->capacity = (pq->capacity) << 1;
    pq->elems = realloc(pq->elems, pq->elemSize * pq->capacity);
    assert(pq->elems);
}

//优先级队列堆空间倍减
static void PQueueReduce(PQUEUE *pq)
{
    if ((pq->capacity >> 1) < pq->initCap)
    {
        return ;
    }
    pq->capacity = (pq->capacity) >> 1;
    pq->elems = realloc(pq->elems, pq->elemSize * pq->capacity);
    assert(pq->elems);
}


//优先级队列初始化
void PQueueNew(PQUEUE *pq, int elemSize, long long initCap, ComplHeapCmp *cmpFn,
ComplHeapFree *freeFn)
{
    assert(0 < elemSize);
    assert(0 < initCap);
    assert(NULL != cmpFn);
    pq->elemSize = elemSize;
    pq->initCap = initCap;
    pq->capacity = initCap;
    pq->cmpFn = cmpFn;
    pq->freeFn = freeFn;
    pq->size = 0;
    pq->elems = malloc(elemSize * initCap);
    assert(NULL != pq->elems);
}

//获取优先级队列元素个数
long long PQueueSize(PQUEUE *pq)
{
    return pq->size;
}

//优先级队列判空，返回值：0--非空，!0--空
int PQueueEmpty(PQUEUE *pq)
{
    return (0 == pq->size);
}
```

```
73
74  //清空优先级队列所有元素
75  void PQueueMakeEmpty(PQUEUE *pq)
76  {
77      if (NULL != pq->freeFn)
78      {
79          int i;
80          for(i = 0; i < pq->size; i++)
81          {
82              pq->freeFn((char *)pq->elems + i * pq->elemSize);
83          }
84      }
85      pq->size = 0;
86      pq->elems = realloc(pq->elems, pq->elemSize * pq->initCap);
87      assert(NULL != pq->elems);
88  }
89
90  //优先级队列销毁
91  void PQueueDispose(PQUEUE *pq)
92  {
93      if (NULL != pq->freeFn)
94      {
95          int i;
96          for(i = 0; i < pq->size; i++)
97          {
98              pq->freeFn((char *)pq->elems + i * pq->elemSize);
99          }
100     }
101     free(pq->elems);
102     pq->elems = NULL;
103     pq->size = 0;
104 }
105
106 //获取当前优先级最大的元素
107 int PQueueGetMax(PQUEUE *pq, void *e)
108 {
109     if (PQueueEmpty(pq) || NULL == e)
110     {
111         return -1;
112     }
113     memcpy(e, pq->elems, pq->elemSize);
114     return 0;
115 }
116
117 //从第pos个节点开始上虑
118 static void heapFilterUp(PQUEUE *pq, long long pos)
119 {
120     long long c = pos; //current node
121     long long p = Parent(c); //parent node
122     void *tmp = malloc(pq->elemSize);
123     assert(NULL != tmp);
124     memcpy(tmp, (char *)pq->elems + c * pq->elemSize, pq->elemSize);
125     void *cAddr, *pAddr;
126     while (ParentValid(c))
127     {
128         cAddr = (char *)pq->elems + c * pq->elemSize;
129         pAddr = (char *)pq->elems + p * pq->elemSize;
130         //父节点的优先级低于待上虑节点的优先级继续上虑
131         if (pq->cmpFn(tmp, pAddr) > 0)
132         {
133             memcpy(cAddr, pAddr, pq->elemSize);
134             c = p;
135             p = Parent(p);
136         }
137         else
138         {
139             break;
140         }
141     }
142     cAddr = (char *)pq->elems + c * pq->elemSize;
143     memcpy(cAddr, tmp, pq->elemSize);
144     free(tmp);
145 }
```

```
146
147 //优先级队列插入关键码e，返回值：0--成功，!0--失败
148 int PQueueInsert(PQUEUE *pq, const void *e)
149 {
150     //当前队列满了
151     if (pq->capacity == pq->size)
152     {
153         return -1;
154         //队列达到扩展上限，插入失败
155         //if ((pq->capacity << 1) < 0)
156         //{
157         //  return -1;
158         //}
159         //else
160         //{
161         //  PQueueGrow(pq);
162         //}
163     }
164     memcpy((char *)pq->elems + pq->size * pq->elemSize, e, pq->elemSize);
165     pq->size ++;
166     heapFilterUp(pq, pq->size - 1);
167     return 0;
168 }
169
170 static void heapFilterDown(PQUEUE *pq, long long begin, long long end)
171 {
172     long long cur = begin; //current node
173     long long left = LChild(cur); //left child position
174     void *tmp = malloc(pq->elemSize);
175     assert(NULL != tmp);
176     memcpy(tmp, pq->elems, pq->elemSize);
177     for (; left < end; cur = left, left = LChild(left))
178     {
179         void *lAddr = (char *)pq->elems + left * pq->elemSize;
180         if (left + 1 < end)
181         {
182             void *rAddr = (char *)pq->elems + (left + 1) * pq->elemSize;
183             if (pq->cmpFn(rAddr, lAddr) > 0)
184             {
185                 lAddr = rAddr;
186                 left ++;
187             }
188         }
189         //待下虑节点的优先级低于两个孩子节点中最大的优先级继续下虑
190         if (pq->cmpFn(lAddr, tmp) > 0)
191         {
192             void *cAddr = (char *)pq->elems + cur * pq->elemSize;
193             memcpy(cAddr, lAddr, pq->elemSize);
194         }
195         else
196         {
197             break;
198         }
199     }
200     void *cAddr = (char *)pq->elems + cur * pq->elemSize;
201     memcpy(cAddr, tmp, pq->elemSize);
202     free(tmp);
203 }
204
205 //优先级队列删除优先级最大的元素
206 int PQueueDeleteMax(PQUEUE *pq)
207 {
208     if (PQueueEmpty(pq))
209     {
210         return -1;
211     }
212     if (NULL != pq->freeFn)
213     {
214         pq->freeFn(pq->elems);
215     }
216     void *last = (char *)pq->elems + (pq->size - 1) * pq->elemSize;
217     if (PQueueSize(pq) > 1)
218     {
```

```
219             memcpy(pq->elems, last, pq->elemSize); //将最后一个元素覆盖到第一个元素
220         }
221         pq->size --;
222         heapFilterDown(pq, 0, pq->size);
223         //队列进行紧缩
224         //if  (pq->size <= ((pq->capacity >> 1) - REDUCTION_THRESHOLD) && pq->capacity >
        pq->initCap)
225         //{
226         //  PQueueReduce(pq);
227         //}
228         return 0;
229     }
230
231     //批量建堆，Floyd建堆算法，时间复杂度O(n)
232     void PQHeapify(PQUEUE *pq)
233     {
234         int i = LastInternal(pq->size);
235         for (; InHeap(pq->size, i); i --)
236         {
237             heapFilterDown(pq, i, pq->size);
238         }
239     }
```