```c
/* SkewHeap.c */
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <assert.h>
#include "SkewHeap.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;
    memcpy(newNode->key, e, keySize);
    return newNode;
}

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

//PQueue初始化
void PQueueNew(PQUEUE *pq, int keySize, SkewHeapCmp *cmpFn, SkewHeapFree *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 ;
    }
    QUEUE nodeQueue;
    QueueNew(&nodeQueue, sizeof(BINNODE *), NULL);
    BINNODE *node = pq->root;
    QueueEn(&nodeQueue, &node);
    while (!QueueEmpty(&nodeQueue))
    {
        QueueDe(&nodeQueue, &node);
        if (NULL != node->lc)
        {
            QueueEn(&nodeQueue, &(node->lc));
        }
        if (NULL != node->rc)
        {
            QueueEn(&nodeQueue, &(node->rc));
        }
        nodeDispose(node, pq->freeFn);
    }
    QueueDispose(&nodeQueue);
    pq->root = NULL;
    pq->size = 0;
}

//获取当前优先级最大的元素
int PQueueGetMax(PQUEUE *pq, void *e)
{
    if (PQueueEmpty(pq) || NULL == e)
    {
        return -1;
    }
    memcpy(e, pq->root->key, pq->keySize);
    return 0;
}

//合并以a和b为根节点的两个斜堆
static BINNODE *merge(PQUEUE *pq, BINNODE *a, BINNODE *b)
{
    if (NULL == a)
    {
        return b;
    }
    if (NULL == b)
    {
        return a;
    }
    if (0 > pq->cmpFn(a->key, b->key))
    {
        BINNODE *tmp = a;
        a = b;
        b = tmp;
    }
    a->rc = merge(pq, a->rc, b);
    a->rc->parent = a;
    //交换以a为根的左右分支
    BINNODE *tmp = a->lc;
    a->lc= a->rc;
    a->rc = tmp;
    return a;
}

//优先级队列插入关键码e，返回值：0--成功，!0--失败
int PQueueInsert(PQUEUE *pq, const void *e)
{
    BINNODE *newNode = nodeNew(pq->keySize, e);
    if (NULL == newNode)
    {
        return -1;
    }
    pq->root = merge(pq, pq->root, newNode);
    pq->root->parent = NULL;
    pq->size ++;
    return 0;
}

//优先级队列删除优先级最大的元素
```

```c
int PQueueDeleteMax(PQUEUE *pq)
{
    if (PQueueEmpty(pq))
    {
        return -1;
    }
    BINNODE *lHeap = pq->root->lc;
    BINNODE *rHeap = pq->root->rc;
    nodeDispose(pq->root, pq->freeFn);
    pq->size --;
    pq->root = merge(pq, lHeap, rHeap);
    if (NULL != pq->root)
    {
        pq->root->parent = NULL;
    }
    return 0;
}
```