Задача: реализовать приоритетную очередь

Приоритетная очередь, это очередь, в которой для каждого элемента указан приоритет. Добавление элемента в очередь возможно с любым приоритетом. При извлечении элемента из очереди, возвращается элемент с наибольшим или с наименьшим приоритетом в зависимости от требований (https://en.wikipedia.org/wiki/Priority_queue)

- Требуется реализовать приоритетную очередь в виде двоичной кучи: (https://en.wikipedia.org/wiki/Binary_heap).
- Извлекаемый элемент должен иметь наименьший приоритет.
- Операции добавления и извлечения элементов должны работать за логарифмическое время.
- При добавлении элемента пользователь указывает приоритет и указатель на данные элемента (void *).
- При извлечении элемента нужно вернуть приоритет и указатель на данные.
- Никаких операций с самими данные производить не нужно - т. е. не нужно пытаться копировать их, или выделять для них память.

Требуется предоставить два файла:

- `priority_queue.h` - интерфейс приоритетной очереди.
- `priority_queue.c` - реализация приоритетной очереди.

Общие требования

Язык реализации - Си в стандарте C99.

В наименовании переменных используется так называемая венгерская нотация, т.е. имя переменой содержит ее тип. Примеры

```
int n_index, n_count;
float f_sum;
int * pn_result;
float * pf_vector;
```

Возвращаемые значения

```
#define STATUS_OK              (0)  /* Successfully completed. */
#define STATUS_ERR_INVALID     (-1)  /* Bad argument value. E.g. null pointer */
#define STATUS_ERR_NOMEM       (-2)  /* No memory to complete operation. */
#define STATUS_ERR_NODATA      (-3)  /* No data to return/process. */
```

Если нужно вернуть значение некоторой величины, то оно возвращается по указателю, переданному в функцию. Пример:

```
int getSomething(..., int * pn_something)
{
    … *pn_something = value;
}
```

Интерфейс приоритетной очереди

```
typedef struct _PrQueue PrQueue;

int prQueueCreate(size_t n_capacity, PrQueue ** pp_queue);
/*  Brief: Create priority queue with specified capacity.
    Input:
        n_capacity    - Queue capacity.
        pp_queue      - Pointer to the variable to fill with new queue pointer.
    Output:
        *pp_queue     - Will contain a pointer to a newly created queue.
    Return:
        STATUS_OK           - Operation completed successfully
        STATUS_ERR_INVALID  - Invalid arguments passed.
        STATUS_ERR_NOMEM    - Not enough memory to allocate the queue.
*/

int prQueuePush(PrQueue * p_queue, int n_priority, void * p_item);
/*  Brief: Push an item to the queue.
    Input:
        p_queue     - A pointer to the priority queue.
        n_priority  - New item priority.
        p_item      - New item data.
    Return:
        STATUS_OK           - Operation completed succesfully
        STATUS_ERR_INVALID  - Invalid arguments passed.
        STATUS_ERR_NOMEM    - The queue capacity exceeded.
*/

int prQueuePop(PrQueue * p_queue, int * pn_priority, void ** pp_item);
/*  Brief: Pop (get and remove) item with minimal priority from the queue.
    Input:
        p_queue      - A priority queue pointer.
        pn_priority  - A pointer to the variable to fill with item priority.
        pp_item      - A pointer to the variable to fill with item data pointer.
    Output:
        *pn_priority - Extracted item priority.
        *pp_item     - Extracted item data pointer.
    Return:
        STATUS_OK           - Operation completed successfully
        STATUS_ERR_INVALID  - Invalid arguments passed.
        STATUS_ERR_NODATA   - There are no items in the queue.
*/

int prQueuePeek(PrQueue * p_queue, int * priority, void ** pp_item);
/*  Brief: Peek item with minimal priority from the queue.
    Same as prQueuePop(...) except that this function does not
    remove item from the queue.
*/
```

```
int prQueueGetCount(PrQueue * p_queue, size_t * pn_count);
/*  Brief: Get number of elements in the queue.
    Input:
        p_queue      - A pointer to the priority queue.
        pn_count     - A pointer to the variable to fill with number of elements.
    Output:
        *pn_count    - Number of elements in the queue.
    Return:
        STATUS_OK           - Operation completed successfully
        STATUS_ERR_INVALID  - Invalid arguments passed.
*/

void prQueueRelease(PrQueue * p_queue);
/*  Brief: Destroy queue and free the memory.
    Input:
        p_queue      - A pointer to the priority queue.
    Return:
        Nothing
*/
```