```
#ifndef _PRIORITYQUEUE_H
#define _PRIORITYQUEUE_H

#include <iostream>
using namespace std;

#include <string>
#include <sstream>

//#include "Array.h"

//-----------------------------------------------------------
//for detecting memory leaks:
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#ifdef _DEBUG
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif // _DEBUG
//-----------------------------------------------------------

//NOTES:
//size_t its typedef to unsigned int
//my implementation to PQ - is an array the when you PUSH an element into him its placing the element with keeping the order from SMALL - TO - BIG
//always to make sure that if a place in array is empty - so put there NULL! - -NOTE : i didnt do it (didnt need to)
//maybe its a good idea to implement dequeue(dual)! - NOTE : i didnt do it (didnt need to)

template <class T>
class PriorityQueue
{
private:
    int m_capacity;//full size of array (also with the empty places)
    T* m_arr; //array of T elements (in size of m_capacity)
    int m_size;//the amount of element in the array.
    int m_top;//the place in the array of the first element to come out (m_top = m_size -1 ) -THATS FOR ORDER SMALL - TO - BIG

public:
    //ctor
    //if len is negative or 0. so it just give values to m_capacity=0 ,m_size=0 ,m_top=-1 , and dont create m_arr. and if the person do "push" so it resize himself resize(2*m_size+1) and do good .
    //if len is positive so m_capacity=len ,m_size=0 ,m_top=-1 , and *yes* - create m_arr in len of m_capacity .
    //NOTE: if person put a negative len so its print a massage to screen
    PriorityQueue(int len = 5) : m_capacity( len >= 0 ? len : 0 ) , m_arr ( len>0 ? (new T[len]) : NULL) , m_size(0) , m_top(-1)
```

```
    {
        if (len < 0 )
            cout << "You must create PQ with a non negetive len - BUT dont worry when you push item it will create a valid PQ for you" << endl;
    }

    //dtor
    ~PriorityQueue()
    {
        delete[] m_arr;
    }

    ////cctor: NOTE:didnt need to!
    //PriorityQueue(const PriorityQueue& other_PQ)
    //{
    //  //like yael in array:
    //  m_capacity = other_PQ.m_capacity;
    //  if (m_capacity > 0)
    //  {
    //      m_arr = new T[m_capacity];
    //      for (int i = 0; i < m_capacity; i++)
    //          m_arr[i] = other_PQ.m_arr[i];
    //  }
    //  else
    //      m_arr = NULL;
    //  //
    //  m_size = other_PQ.m_size;
    //  m_top = other_PQ.m_top;
    //}

    const T& top() const
    {
        return m_arr[m_top];
    }

    bool empty() const
    {
        if (m_size == 0)
            return true;
        return false;
    }

    const int size() const
    {
        return m_size;
    }

    //push element with keeping the order in m_arr from SMALL - TO - BIG . update m_size ,m_top.
    //if m_arr is full , so resize it! and do the same .
    void push(const T& element)
    {
```

```
        if (full())
            resize( (m_size * 2 + 1) );
        int index_to_compare = m_top;
        while (m_arr[index_to_compare] > element && index_to_compare >= 0)
        {
            m_arr[index_to_compare + 1] = m_arr[index_to_compare];
            index_to_compare--;
        }
        m_arr[(index_to_compare + 1)] = element;
        m_size++;
        m_top++;
    }

    //pop the element and update m_top , m_size .
    const T& pop()
    {
        T tmp_T = m_arr[m_top];
        m_arr[m_top] = NULL;
        m_top--;
        m_size--;
        return tmp_T;
    }

    //PRINT FROM BIG - TO - SMALL
    friend ostream& operator<<(ostream& os, const PriorityQueue<T>& PQ)
    {
        if (!PQ.empty())
        {
            os << "{";
            for (int i = PQ.m_top; i > 0; i--)
                os << PQ.m_arr[i] << ",";
            os << PQ.m_arr[0] << "}";
        }
        else
            os << "PQ is empty!";
        return os;
    }

    //-------------------------------------------------------------------
    ------------------------------------------------------------------

    //Extras:

    //making the queue in new *bigger* size. return true if succeeded ,
      false if not. |?need also to smaller size?-didnt do it (didnt need
      to)|
    const bool resize(int new_size)
    {
        //if m_capacity == 0 (already) , so "there is not a queue" at
          all , so create a new one (without copy because there is nothing
          to copy) | the "new one" will be in exactly the same address like
           the old
        if (!m_capacity)
```

```
        {
            *this = PriorityQueue(new_size);;
            return true;
        }

        //if new size == to m_size (already) ,so do nothing.
        if (m_size == new_size)
            return true;

        //if m_capacity < new_size , so we need to create new m_arr , copy all the data in m_arr , and also change (just) the member m_capacity (m_size and m_top dont change) | the "new m_arr" will be in exactly the same address like the old
        if (m_capacity < new_size)
        {
            int new_m_capacity = new_size;
            T* new_m_arr = new T[new_m_capacity];
            int i = 0;
            for (; i < m_size; i++)
                new_m_arr[i] = m_arr[i];
            delete[] m_arr;
            m_capacity = new_m_capacity;
            m_arr = new_m_arr;
            return true;
        }
        return false;
    }

    const bool full() const
    {
        if (m_size == m_capacity)
            return true;
        return false;
    }

    const T* get_m_arr() const
    {
        return m_arr;
    }

    //for deep copy
    const PriorityQueue<T>& operator=(const PriorityQueue<T>& other)
    {
        if (&other == this)
            return *this;
        if (m_capacity != other.m_capacity)
        {
            if (m_arr)
                delete[] m_arr;
            m_capacity = other.m_capacity;

            if (m_capacity > 0)
                m_arr = new T[m_capacity];
```

```
        }
        m_size = other.m_size;
        m_top = other.m_top;
        // deep copy
        for (int i = 0; i < m_capacity; i++)
            m_arr[i] = other.m_arr[i];
        return *this;
    }

};//END OF CLASS PriorityQueue

//PRINT FROM SMALL - TO - BIG
//why not inline?
template<typename T>
void print_reversed_queue(const PriorityQueue<T>& PQ)
{
    if (!PQ.empty())
    {
        cout << "{";
        for (int i = 0; i < PQ.size(); i++)
            cout << PQ.get_m_arr()[i] << ",";
        cout << "}";
    }
    else
        cout << "PQ is empty!";
}

#endif // !PriorityQueue

//NOTE:
//PriorityQueue is implemented in PriorityQueue header . ********!!! ITS BECAUSE ITS CLASS TEMPLATE !!!*********


//what i need to do :
//1)printing of array of pq (i want it to print till size of array not capacity) - its yael solution - i didnt changed it!
//2)to fix if some one create pq with len of 0 (or negative len) - fix it or in ctor /in resize /in push - to do resize(2*m_size+1) (because if size is 0 so size*2 is also 0)
//ANSWER FOR 2) : if len is negative or 0. so it just give values to m_capacity=0 ,m_size=0 ,m_top=-1 , and dont create m_arr. and if the person do "push" so it resize himself resize(2*m_size+1) and do good . and also print a message to screen
//                if len is positive so m_capacity=len ,m_size=0 ,m_top=-1 , and *yes* - create m_arr in len of m_capacity

//4)all the size_t maybe to change it to int  - NOTE - i changed it to int because m_top need to be int (it need to be at first m_top = -1)
//5)all the memset or memcpy - dont work good so i leave it (at first i did a for loop but it doesnt matter)

```

```
//6)to check the whole project (with worst case) - DONE!
//7)clean this project from useless line and comments!! - DONE!
//8)end this project (do a word document)!!
```