```
#ifndef _ARRAY_H_
#define _ARRAY_H_

#include <string>
#include <sstream>
using namespace std;

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

//-------------------------------------------------------------------------------
class outOfRange : public exception {
private:
    string m_s;
public:
    outOfRange(string s = "out of range exception") : m_s(s) {};
    virtual const char* what() const {
        return m_s.c_str();
    }
};
//-------------------------------------------------------------------------------

template <class T>
class Array
{
private:
    void check(size_t ndx) const
    {
        if (ndx < 0 || ndx >= m_len)
        {
            stringstream ss;
            ss << "Exception: Invalid index " << ndx << " int array of length " << m_len << endl;
#define STRINGSTREAM
            //#define EXCEPTION_CLASS
#ifdef STRINGSTREAM
            throw(ss.str());
#elif defined(EXCEPTION_CLASS)
            outOfRange oof(ss.str());
            throw(oof);
#endif
        }
```

```
    };
    size_t  m_len;   //size_t is typedef to unsigned int
    T* m_data;

    //-------------------------------------------------------------
public:

    Array(size_t len = 10) : m_len(len), m_data(len ? (new T[len]) : NULL)
    {
        //if (len)
        //  memset(m_data, 0, len*sizeof(T));                       //NOT GOOD!!!!!!!!!!!!!!!!!!!!!
    }
    ~Array() { delete[] m_data; }
    size_t len() const { return m_len; }
    T& operator[](size_t i) { check(i);  return m_data[i]; }
    const T& operator[](size_t i) const { check(i);  return m_data[i]; }

    Array(const Array& other)
    {
        m_len = other.m_len;
        if (m_len > 0)
        {
            m_data = new T[m_len];
            for (int i = 0; i < m_len; i++)
                m_data[i] = other.m_data[i];
        }
        else
            m_data = NULL;
    }
    Array& operator=(const Array& other)
    {
        if (&other == this)
            return *this;
        if (m_len != other.m_len)
        {
            if (m_data)
                delete[] m_data;
            m_len = other.m_len;

            if (m_len > 0)
                m_data = new T[m_len];
        }
        for (int i = 0; i < m_len; i++)
            m_data[i] = other.m_data[i];
        return *this;
    }
    bool operator<(const Array& other)
    {
        return m_len < other.m_len;
    }

    bool operator>(const Array& other)
```

```
    {
        return m_len > other.m_len;
    }

    friend ostream& operator<<(ostream& os, const Array<T>& a) //changed it from yael - from " const Array & a " , to " const Array<T>& a " . and i put this function implementation out from the class
    {
        for (int i = 0; i < a.len(); i++)
            os << a.m_data[i] << " ";                                //changed it from yael - from " a.[i] ", to , " a.m_data[i] "
        return os;
    }

    //friend ostream& operator<<(ostream& os, const Array<T>& a);     //changed it from yael - from " const Array & a " , to " const Array<T>& a " . and i put this function implementation out from the class

};

//template <class T>
//ostream& operator<<(ostream& os, const Array<T>& a)            //changed it from yael
//{
//  for (int i = 0; i < a.len(); i++)
//      os << a.m_data[i] << " ";                               //changed it from yael
//  return os;
//}

#endif

//NOTE:
//Array is implemented in Array header (yael did it) . ********!!! ITS BECAUSE ITS CLASS TEMPLATE !!!*********

//todo list
//1)printing an array - (i want it to print till size of array not capacity)
```