

```

1  #ifndef _ARRAY_H_
2  #define _ARRAY_H_
3
4  #include <string>
5  #include <sstream>
6  using namespace std;
7
8  //-----
9  //for detecting memory leaks:
10 #define _CRTDBG_MAP_ALLOC
11 #include <crtdbg.h>
12 #ifdef _DEBUG
13 #ifndef DBG_NEW
14 #define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
15 #define new DBG_NEW
16 #endif
17 #endif // _DEBUG
18 //-----
19
20 //-----
21 class outOfRange : public exception {
22 private:
23     string m_s;
24 public:
25     outOfRange(string s = "out of range exception") : m_s(s) {};
26     virtual const char* what() const {
27         return m_s.c_str();
28     }
29 };
30 //-----
31
32 template <class T>
33 class Array
34 {
35 private:
36     void check(size_t ndx) const
37     {
38         if (ndx < 0 || ndx >= m_len)
39         {
40             stringstream ss;
41             ss << "Exception: Invalid index " << ndx << " int array of
length " << m_len << endl;
42 #define STRINGSTREAM
43             //#define EXCEPTION_CLASS
44 #ifdef STRINGSTREAM
45                 throw(ss.str());
46 #elif defined(EXCEPTION_CLASS)
47                 outOfRange oof(ss.str());
48                 throw(oof);
49 #endif
50         }

```

```

51     };
52     size_t m_len;    //size_t is typedef to unsigned int
53     T* m_data;
54
55     //-----
56 public:
57
58     Array(size_t len = 10) : m_len(len), m_data(len ? (new T[len]) : NULL)
59     {
60         //if (len)
61         //    memset(m_data, 0, len*sizeof(T));           //NOT ↗
62         //    GOOD!!!!!!!!!!!!!!!!!!!!!!
63     }
64     ~Array() { delete[] m_data; }
65     size_t len() const { return m_len; }
66     T& operator[](size_t i) { check(i); return m_data[i]; }
67     const T& operator[](size_t i) const { check(i); return m_data[i]; }
68
69     Array(const Array& other)
70     {
71         m_len = other.m_len;
72         if (m_len > 0)
73         {
74             m_data = new T[m_len];
75             for (int i = 0; i < m_len; i++)
76                 m_data[i] = other.m_data[i];
77         }
78         else
79             m_data = NULL;
80     }
81     Array& operator=(const Array& other)
82     {
83         if (&other == this)
84             return *this;
85         if (m_len != other.m_len)
86         {
87             if (m_data)
88                 delete[] m_data;
89             m_len = other.m_len;
90
91             if (m_len > 0)
92                 m_data = new T[m_len];
93         }
94         for (int i = 0; i < m_len; i++)
95             m_data[i] = other.m_data[i];
96         return *this;
97     }
98     bool operator<(const Array& other)
99     {
100         return m_len < other.m_len;
101     }
102     bool operator>(const Array& other)

```

```
103     {
104         return m_len > other.m_len;
105     }
106
107     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
108     {
109         for (int i = 0; i < a.len(); i++)
110             os << a.m_data[i] << " "; // ↗
        changed it from yael - from " a.[i] ", to , " a.m_data[i] "
111         return os;
112     }
113
114     //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
115
116 };
117
118 //template <class T>
119 //ostream& operator<<(ostream& os, const Array<T>& a) //changed ↗
        it from yael
120 //{
121 //    for (int i = 0; i < a.len(); i++)
122 //        os << a.m_data[i] << " "; //changed ↗
        it from yael
123 //    return os;
124 //}
125
126 #endif
127
128 //NOTE:
129 //Array is implemented in Array header (yael did it) . *****!!! ITS ↗
        BECAUSE ITS CLASS TEMPLATE !!!*****
130
131 //todo list
132 //1)printing an array - (i want it to print till size of array not ↗
        capacity)
```