

**Swinburne University of Technology**  
*Faculty of Science, Engineering and Technology*

**ASSIGNMENT COVER SHEET**

---

**Subject Code:** COS30008  
**Subject Title:** Data Structures and Patterns  
**Assignment number and title:** 2, Indexers, Method Overriding, and Lambdas  
**Due date:** April 7, 2022, 14:30  
**Lecturer:** Dr. Markus Lumpe

---

**Your name:** \_\_\_\_\_ **Your student id:** \_\_\_\_\_

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30

---

Marker's comments:

Problem	Marks	Obtained
1	48	
2	30+10= 40	
3	58	
Total	146	

---

**Extension certification:**

This assignment has been given an extension and is now due on \_\_\_\_\_

Signature of Convener: \_\_\_\_\_

```
1 // Problem Set 2, 2024
2
3 #include <stdexcept>
4 #include "IntVector.h"
5
6 IntVector::IntVector(const int aArrayOfIntegers[], size_t
    aNumberOfElements) :
7     // member initializer
8     fNumberOfElements(aNumberOfElements)
9 {
10     // creates a dynamic array of int
11     fElements = new int[fNumberOfElements];
12
13     for (size_t i = 0; i < aNumberOfElements; i++)
14     {
15         fElements[i] = aArrayOfIntegers[i];
16     }
17 }
18
19 IntVector::~IntVector()
20 {
21     // releases memory of the dynamic array
22     delete[] fElements;
23 }
24
25 size_t IntVector::size() const
26 {
27     return fNumberOfElements;
28 }
29
30 const int IntVector::get(size_t aIndex) const
31 {
32     // reuse operator[] to checks index
33     return (*this)[aIndex];
34 }
35
36 void IntVector::swap(size_t aSourceIndex, size_t aTargetIndex)
37 {
38     // checks if indices are in range
39     // we could also reuse operator[] to check each element
40     if ((aSourceIndex >= fNumberOfElements) ||
41         (aTargetIndex >= fNumberOfElements))
42     {
43         throw std::out_of_range("Illegal vector indices!!");
44     }
45
46     int lSourceElement = fElements[aSourceIndex];
47     fElements[aSourceIndex] = fElements[aTargetIndex];
48     fElements[aTargetIndex] = lSourceElement;
```

```
49 }
50
51 const int IntVector::operator[](size_t aIndex) const
52 {
53     // checks if index is in range
54     if (aIndex >= fNumberOfElements)
55     {
56         throw std::out_of_range("Illegal vector index!");
57     }
58
59     return fElements[aIndex];
60 }
```

```
1 // Problem Set 2, 2024
2
3 #include "SortableIntVector.h"
4
5 SortableIntVector::SortableIntVector(const int aArrayOfIntegers[], size_t ↗
    aNumberOfElements) :
6     // calls super class constructor
7     IntVector(aArrayOfIntegers, aNumberOfElements)
8 { }
9
10 void SortableIntVector::sort(Comparable aOrderFunction)
11 {
12     // only calls the getter once
13     size_t lSize = size();
14
15     // sorts in INCREASING order
16     for (size_t i = 0; i < lSize - 1; i++) // outer loop
17     {
18         for (size_t j = 0; j < lSize - 1 - i; j++) // inner loop
19         {
20             // compares adjacent elements and swaps if the former has ↗
21             // bigger value
22             if (!aOrderFunction((*this)[j], (*this)[j + 1]))
23             {
24                 swap(j, j + 1);
25             }
26         }
27     }
```

```
1 // Problem Set 2, 2024
2
3 #include "ShakerSortableIntVector.h"
4
5 ShakerSortableIntVector::ShakerSortableIntVector(const int aArrayOfIntegers ↗
    [], size_t aNumberOfElements) :
6     // calls super class constructor
7     SortableIntVector(aArrayOfIntegers, aNumberOfElements)
8 { }
9
10 void ShakerSortableIntVector::sort(Comparable aOrderFunction)
11 {
12     size_t lStart = 0;
13     size_t lEnd = size() - 1;
14
15     // sorts in DECREASING order without a "sorted" flag
16     while (lStart < lEnd)
17     {
18         // forward loop from left to right
19         // bubbles the smallest value to the end of the array
20         for (size_t i = lStart; i < lEnd; i++)
21         {
22             // compares and swaps if the former element is smaller
23             if (!aOrderFunction((*this)[i + 1], (*this)[i]))
24             {
25                 swap(i, i + 1);
26             }
27         }
28
29         // moves the end point back by one
30         lEnd--;
31
32         // backward loop from right to left
33         // bubbles the largest value to the start of the array
34         for (size_t i = lEnd; i > lStart; i--)
35         {
36             // compares and swaps if the latter element is bigger
37             if (!aOrderFunction((*this)[i], (*this)[i - 1]))
38             {
39                 swap(i - 1, i);
40             }
41         }
42
43         // moves the start point ahead by one
44         lStart++;
45     }
46 }
```

```
1
2 // Problem Set 2, 2022
3
4 #include <iostream>
5 #include <stdexcept>
6
7 using namespace std;
8
9 #define P1
10 #define P2
11 #define P3
12
13 #ifdef P1
14
15 #include "IntVector.h"
16
17 void runP1()
18 {
19     int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
20     size_t lArrayLength = sizeof(lArray) / sizeof(int);
21
22     IntVector lVector( lArray, lArrayLength );
23
24     cout << "Test range check:" << endl;
25
26     try
27     {
28         int lValue = lVector[lArrayLength];
29
30         cerr << "Error, you should not see " << lValue << " here!" <<      ↗
31             endl;
32     }
33     catch (out_of_range e)
34     {
35         cerr << "Properly caught error: " << e.what() << endl;
36     }
37     catch (...)
38     {
39         cerr << "This message must not be printed!" << endl;
40     }
41
42     cout << "Test swap:" << endl;
43
44     try
45     {
46         cout << "lVector[3] = " << lVector[3] << endl;
47         cout << "lVector[6] = " << lVector[6] << endl;
48
49         lVector.swap( 3, 6 );
```

```
49
50     cout << "lVector.get( 3 ) = " << lVector.get( 3 ) << endl;
51     cout << "lVector.get( 6 ) = " << lVector.get( 6 ) << endl;
52
53     lVector.swap( 5, 20 );
54
55     cerr << "Error, you should not see this message!" << endl;
56 }
57 catch (out_of_range e)
58 {
59     cerr << "Properly caught error: " << e.what() << endl;
60 }
61 catch (...)
62 {
63     cerr << "Error, this message must not be printed!" << endl;
64 }
65 }
66
67 #endif
68
69 #ifdef P2
70
71 #include "SortableIntVector.h"
72
73 void runP2()
74 {
75     int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
76     size_t lArrayLength = sizeof(lArray) / sizeof(int);
77
78     SortableIntVector lVector( lArray, lArrayLength );
79
80     cout << "Bubble Sort:" << endl;
81
82     cout << "Before sorting:" << endl;
83
84     for ( size_t i = 0; i < lVector.size(); i++ )
85     {
86         cout << lVector[i] << ' ';
87     }
88
89     cout << endl;
90
91     // Use a lambda expression here that orders integers in increasing order. ➤
92     // The lambda expression does not capture any variables or throws any ➤
93     // exceptions.
94     // It has to return a bool value.
95     lVector.sort([] (int aLHS, int aRHS) -> bool
96     {
```

```
96         return aLHS <= aRHS;
97     });
98
99     cout << "After sorting:" << endl;
100
101     for ( size_t i = 0; i < lVector.size(); i++ )
102     {
103         cout << lVector[i] << ' ';
104     }
105
106     cout << endl;
107 }
108
109 #endif
110
111 #ifdef P3
112
113 #include "ShakerSortableIntVector.h"
114
115 void runP3()
116 {
117     int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
118     size_t lArrayLength = sizeof(lArray) / sizeof(int);
119
120     ShakerSortableIntVector lVector( lArray, lArrayLength );
121
122     cout << "Cocktail Shaker Sort:" << endl;
123
124     cout << "Before sorting:" << endl;
125
126     for ( size_t i = 0; i < lVector.size(); i++ )
127     {
128         cout << lVector[i] << ' ';
129     }
130
131     cout << endl;
132
133     // sort in decreasing order
134     lVector.sort();
135
136     cout << "After sorting:" << endl;
137
138     for ( size_t i = 0; i < lVector.size(); i++ )
139     {
140         cout << lVector[i] << ' ';
141     }
142
143     cout << endl;
144 }
```



```
145
146 #endif
147
148 int main()
149 {
150     #ifdef P1
151
152         runP1();
153
154     #endif
155
156     #ifdef P2
157
158         runP2();
159
160     #endif
161
162     #ifdef P3
163
164         runP3();
165
166     #endif
167
168     return 0;
169 }
170
```