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: Nguyen Thi Thanh Minh **Your student id:** 104169617

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
			X	X							

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:___________________________

```cpp
// Problem Set 2, 2024

#include <stdexcept>
#include "IntVector.h"

IntVector::IntVector(const int aArrayOfIntegers[], size_t 
  aNumberOfElements) :
    // member initializer
    fNumberOfElements(aNumberOfElements)
{
    // creates a dynamic array of int
    fElements = new int[fNumberOfElements];

    for (size_t i = 0; i < aNumberOfElements; i++)
    {
        fElements[i] = aArrayOfIntegers[i];
    }
}

IntVector::~IntVector()
{
    // releases memory of the dynamic array
    delete[] fElements;
}

size_t IntVector::size() const
{
    return fNumberOfElements;
}

const int IntVector::get(size_t aIndex) const
{
    // reuse operator[] to checks index
    return (*this)[aIndex];
}

void IntVector::swap(size_t aSourceIndex, size_t aTargetIndex)
{
    // checks if indices are in range
    // we could also reuse operator[] to check each element
    if ((aSourceIndex >= fNumberOfElements) ||
        (aTargetIndex >= fNumberOfElements))
    {
        throw std::out_of_range("Illegal vector indices!!");
    }

    int lSourceElement = fElements[aSourceIndex];
    fElements[aSourceIndex] = fElements[aTargetIndex];
    fElements[aTargetIndex] = lSourceElement;
}

const int IntVector::operator[](size_t aIndex) const
{
    // checks if index is in range
    if (aIndex >= fNumberOfElements)
    {
        throw std::out_of_range("Illegal vector index!");
    }

    return fElements[aIndex];
}
```

```cpp
// Problem Set 2, 2024

#include "SortableIntVector.h"

SortableIntVector::SortableIntVector(const int aArrayOfIntegers[], size_t
  aNumberOfElements) :
    // calls super class constructor
    IntVector(aArrayOfIntegers, aNumberOfElements)
{ }

void SortableIntVector::sort(Comparable aOrderFunction)
{
    // only calls the getter once
    size_t lSize = size();

    // sorts in INCREASING order
    for (size_t i = 0; i < lSize - 1; i++)  // outer loop
    {
        for (size_t j = 0; j < lSize - 1 - i; j++)  // inner loop
        {
            // compares adjacent elements and swaps if the former has
              bigger value
            if (!aOrderFunction((*this)[j], (*this)[j + 1]))
            {
                swap(j, j + 1);
            }
        }
    }
}
```

```cpp
// Problem Set 2, 2024

#include "ShakerSortableIntVector.h"

ShakerSortableIntVector::ShakerSortableIntVector(const int aArrayOfIntegers
  [], size_t aNumberOfElements) :
    // calls super class constructor
    SortableIntVector(aArrayOfIntegers, aNumberOfElements)
{ }

void ShakerSortableIntVector::sort(Comparable aOrderFunction)
{
    size_t lStart = 0;
    size_t lEnd = size() - 1;

    // sorts in DECREASING order without a "sorted" flag
    while (lStart < lEnd)
    {
        // forward loop from left to right
        // bubbles the smallest value to the end of the array
        for (size_t i = lStart; i < lEnd; i++)
        {
            // compares and swaps if the former element is smaller
            if (!aOrderFunction((*this)[i + 1], (*this)[i]))
            {
                swap(i, i + 1);
            }
        }

        // moves the end point back by one
        lEnd--;

        // backward loop from right to left
        // bubbles the largest value to the start of the array
        for (size_t i = lEnd; i > lStart; i--)
        {
            // compares and swaps if the latter element is bigger
            if (!aOrderFunction((*this)[i], (*this)[i - 1]))
            {
                swap(i - 1, i);
            }
        }

        // moves the start point ahead by one
        lStart++;
    }
}
```

```

// Problem Set 2, 2022

#include <iostream>
#include <stdexcept>

using namespace std;

#define P1
#define P2
#define P3

#ifdef P1

#include "IntVector.h"

void runP1()
{
    int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
    size_t lArrayLength = sizeof(lArray) / sizeof(int);

    IntVector lVector( lArray, lArrayLength );

    cout << "Test range check:" << endl;

    try
    {
        int lValue = lVector[lArrayLength];

        cerr << "Error, you should not see " << lValue << " here!" <<
          endl;
    }
    catch (out_of_range e)
    {
        cerr << "Properly caught error: " << e.what() << endl;
    }
    catch (...)
    {
        cerr << "This message must not be printed!" << endl;
    }

    cout << "Test swap:" << endl;

    try
    {
        cout << "lVector[3] = " << lVector[3] << endl;
        cout << "lVector[6] = " << lVector[6] << endl;

        lVector.swap( 3, 6 );

        cout << "lVector.get( 3 ) = " << lVector.get( 3 ) << endl;
        cout << "lVector.get( 6 ) = " << lVector.get( 6 ) << endl;

        lVector.swap( 5, 20 );

        cerr << "Error, you should not see this message!" << endl;
    }
    catch (out_of_range e)
    {
        cerr << "Properly caught error: " << e.what() << endl;
    }
    catch (...)
    {
        cerr << "Error, this message must not be printed!" << endl;
    }
}

#endif

#ifdef P2

#include "SortableIntVector.h"

void runP2()
{
    int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
    size_t lArrayLength = sizeof(lArray) / sizeof(int);

    SortableIntVector lVector( lArray, lArrayLength );

    cout << "Bubble Sort:" << endl;

    cout << "Before sorting:" << endl;

    for ( size_t i = 0; i < lVector.size(); i++ )
    {
        cout << lVector[i] << ' ';
    }

    cout << endl;

    // Use a lambda expression here that orders integers in increasing order.
    // The lambda expression does not capture any variables of throws any exceptions.
    // It has to return a bool value.
    lVector.sort([] (int aLHS, int aRHS) -> bool
        {
            return aLHS <= aRHS;
        });

    cout << "After sorting:" << endl;

    for ( size_t i = 0; i < lVector.size(); i++ )
    {
        cout << lVector[i] << ' ';
    }

    cout << endl;
}

#endif

#ifdef P3

#include "ShakerSortableIntVector.h"

void runP3()
{
    int lArray[] = { 34, 65, 890, 86, 16, 218, 20, 49, 2, 29 };
    size_t lArrayLength = sizeof(lArray) / sizeof(int);

    ShakerSortableIntVector lVector( lArray, lArrayLength );

    cout << "Cocktail Shaker Sort:" << endl;

    cout << "Before sorting:" << endl;

    for ( size_t i = 0; i < lVector.size(); i++ )
    {
        cout << lVector[i] << ' ';
    }

    cout << endl;

    // sort in decreasing order
    lVector.sort();

    cout << "After sorting:" << endl;

    for ( size_t i = 0; i < lVector.size(); i++ )
    {
        cout << lVector[i] << ' ';
    }

    cout << endl;
}
```

```

#endif

int main()
{
#ifdef P1

    runP1();

#endif

#ifdef P2

    runP2();

#endif

#ifdef P3

    runP3();

#endif

    return 0;
}

```