COS30008 Semester 1, 2024 Dr. James Jackson

# 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:** Friday, March 8, 2024, 23:59

**Lecturer:** Dr. James Jackson

## Your name: Nguyen Gia Binh Your student id: 104219428

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | Sat 10:00 |
|  |  |  |  |  |  |  |  |  |  | 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:

1

Main\_PS2.cpp

// 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 a, int b)

{

return a <= b;

});

//lVector.sort( /\* lambda expression \*/ );

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;

}

IntVector.cpp

#include "IntVector.h"

#include <stdexcept>

using namespace std;

IntVector::IntVector(const int aArrayOfIntegers[], size\_t aNumberOfElements)

{

fNumberOfElements = aNumberOfElements;

fElements = new int[fNumberOfElements];

for (size\_t i = 0; i < fNumberOfElements; i++)

{

fElements[i] = aArrayOfIntegers[i];

}

}

IntVector:: ~IntVector() {

delete[] fElements;

}

size\_t IntVector::size() const {

return fNumberOfElements;

}

const int IntVector::get(size\_t aIndex) const {

return (\*this)[aIndex];

}

void IntVector::swap(size\_t aSourceIndex, size\_t aTargetIndex) {

if (aSourceIndex < fNumberOfElements && aTargetIndex < fNumberOfElements) {

size\_t lBuffer = fElements[aSourceIndex];

fElements[aSourceIndex] = fElements[aTargetIndex];

fElements[aTargetIndex] = lBuffer;

}

else

{

throw out\_of\_range("Illegal vector indices");

}

}

const int IntVector::operator[](size\_t aIndex) const {

if (aIndex < fNumberOfElements) {

return fElements[aIndex];

}

else {

throw out\_of\_range("Index out of range");

}

}

SortableIntVector.cpp

#include "SortableIntVector.h"

using namespace std;

SortableIntVector::SortableIntVector(const int aArrayOfIntegers[], size\_t aNumberOfElements) : IntVector(aArrayOfIntegers, aNumberOfElements)

{

}

void SortableIntVector::sort(Comparable aOrderFunction)

{

size\_t lArraylength = size();

for (size\_t i = 0; i < lArraylength; i++)

{

for (size\_t j = 0; j < lArraylength - i - 1; j++)

{

if (aOrderFunction((\*this)[j + 1], (\*this)[j]))

{

swap(j, j + 1);

}

}

}

}

ShakerSortableIntVector.cpp

#include"ShakerSortableIntVector.h"

using namespace std;

ShakerSortableIntVector::ShakerSortableIntVector(const int aArrayOfIntegers[], size\_t aNumberOfElements) : SortableIntVector(aArrayOfIntegers, aNumberOfElements)

{

}

void ShakerSortableIntVector::sort(Comparable aOrderFunction)

{

size\_t lLeft = 0;

size\_t lRight = size() - 1;

while (lLeft < lRight)

{

for (size\_t i = lLeft; i < lRight; i++)

{

if (!aOrderFunction(get(i), get(i + 1)))

{

swap(i, i + 1);

}

}

lRight--;

for (size\_t i = lRight; i > lLeft; i--)

{

if (!aOrderFunction(get(i - 1), get(i)))

{

swap(i - 1, i);

}

}

lLeft++;

}

}