COS30008 Semester 1, 2022 Dr. Markus Lumpe

# 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:** October 18, 2022, 14:30

**Lecturer:** Dr. Markus Lumpe

**Your name:**

## TRAN QUOC DUNG

**Your student id:**

## 103803891

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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**

**Problem 1:**

**File: IntVector.h**

#pragma once

#include <cstddef>

#include <iostream>

class IntVector

{

private:

int\* fElements;

size\_t fNumberOfElements;

public:

// Constructor: copy argument array

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

// Destructor: release memory

// Destructor is virtual to allow inheritance

virtual ~IntVector();

// size getter

size\_t size() const;

// element getter

const int get(size\_t aIndex) const;

// swap two elements within the vector

void swap(size\_t aSourceIndex, size\_t aTargetIndex);

// indexer

const int operator[](size\_t aIndex) const;

};

**File: IntVector.cpp**

#pragma once

#include "IntVector.h"

#include <iostream>

#include <cstddef>

using namespace std;

//constructor

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];

}

}

// Destructor: release memory

// Destructor is virtual to allow inheritance

IntVector::~IntVector()

{

delete[] fElements;

}

// size getter

size\_t IntVector::size() const

{

return fNumberOfElements;

}

// element getter

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

{

return operator[](aIndex);

}

// swap two elements within the vector

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

{

size\_t num = get(aSourceIndex);

fElements[aSourceIndex] = get(aTargetIndex);

fElements[aTargetIndex] = num;

}

// indexer

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

{

return fElements[aIndex];

}

**Problem 2:**

**File: SortableIntVector.h**

#pragma once

#include "IntVector.h"

#include <functional>

using namespace std;

using Comparable = function<bool(int, int)>;

class SortableIntVector : public IntVector

{

public:

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

void sort (Comparable aOrderFunction);

};

**File: SortableIntVector.cpp**

#pragma once

#include "SortableIntVector.h"

#include <functional>

using namespace std;

using Comparable = function <bool(int, int)>;

//constructor

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

void SortableIntVector::sort(Comparable aOrderFunction) {

int i = 1;

for (i = 0; i < size(); i++)

{

for(int j = i + 1; j < size(); j++)

{

if (aOrderFunction(get(j), get(i)) == true) { swap(j, i); }

}

}

}

**Problem 3:**

**File: ShakerSortableIntVector.h**

#pragma once

#include "SortableIntVector.h"

#include <functional>

class ShakerSortableIntVector : public SortableIntVector

{

public:

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

void sort (Comparable aOrderFunction) ;

};

**File: ShakerSortableIntVector.cpp**

#pragma once

#include "ShakerSortableIntVector.h"

#include <functional>

using namespace std;

//constructor

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

void ShakerSortableIntVector::sort(Comparable aOrderFunction) {

int i = 1;

for (i = 0; i < size(); i++)

{

for (int j = i + 1; j < size(); j++)

{

if (aOrderFunction(get(j), get(i)) == true) { swap(j, i); }

}

}

}

**File: 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: " << "Illegal vector index" << 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: " << "Illegal vector index" << 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 aLeft, int aRight) {return (aLeft <= aRight); });

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([](int aLeft, int aRight) {return (aLeft >= aRight); });

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

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

{

cout << lVector[i] << ' ';

}

cout << endl;

}

#endif

int main()

{

#ifdef P1

cout << "\tProblem 1:" << endl;

runP1();

cout << "\n" << endl;

#endif

#ifdef P2

cout << "\tProblem 2:" << endl;

runP2();

cout << "\n" << endl;

#endif

#ifdef P3

cout << "\tProblem 3:" << endl;

runP3();

cout << "\n" << endl;

#endif

return 0;

}