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 21, 2024, 3:59
Lecturer: Dr. Ky Trung Pham

StudentName: Lau Ngoc Quyen
StudentID: 104198996

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										

Marker's comments:

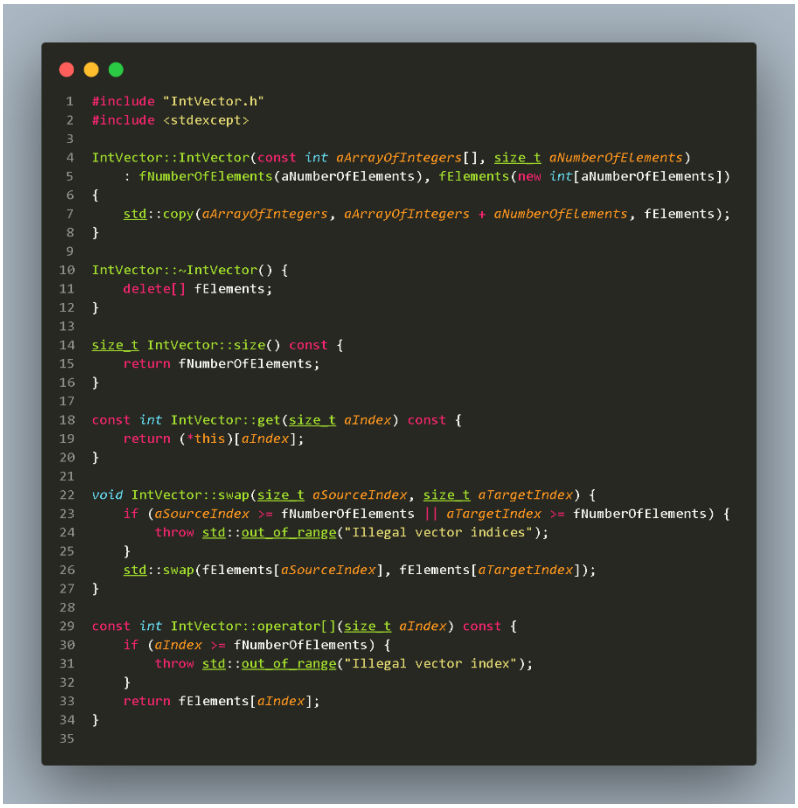
Problem	Marks	Obtained
1	48	
2	30+10= 40	
3	58	
Total	146	

Extension certification:

This assignment has been given an extension and is now due on 21th October 2024

Signature of Convener:_____

1. InVector.cpp



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

IntVector::IntVector(const int aArrayOfIntegers[], size_t aNumberOfElements)
    : fNumberOfElements(aNumberOfElements), fElements(new int[aNumberOfElements])
{
    std::copy(aArrayOfIntegers, aArrayOfIntegers + aNumberOfElements, fElements);
}

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) {
        throw std::out_of_range("Illegal vector indices");
    }
    std::swap(fElements[aSourceIndex], fElements[aTargetIndex]);
}

const int IntVector::operator[](size_t aIndex) const {
    if (aIndex >= fNumberOfElements) {
        throw std::out_of_range("Illegal vector index");
    }
    return fElements[aIndex];
}

```

2. Main.cpp

```cpp
// Main.cpp
#include <iostream>
#include <stdexcept>
#include "IntVector.h"
#include "SortableIntVector.h"
#include "ShakerSortableIntVector.h"

using namespace std;

//#define P1
//#define P2
//#define P3

#ifdef P1
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];  // Out of range access
        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);  // Out of range access
        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
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.
    lVector.sort([](int a, int b) { return a < b; }); // Ascending order
    cout << "After sorting:" << endl;
    for (size_t i = 0; i < lVector.size(); i++) {
        cout << lVector[i] << ' ';
    }
    cout << endl;
}
#endif

#ifdef P3
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 a, int b) { return a > b; }); // Descending order
    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;
}
```

3. SortableIntVector.cpp

```cpp
#include "SortableIntVector.h"

SortableIntVector::SortableIntVector(const int aArrayOfIntegers[], size_t aNumberOfElements)
    : IntVector(aArrayOfIntegers, aNumberOfElements) {}

void SortableIntVector::sort(Comparable aOrderFunction) {
    for (size_t i = 0; i < size() - 1; i++) {
        for (size_t j = 0; j < size() - 1 - i; j++) {
            if (!aOrderFunction(get(j), get(j + 1))) {
                swap(j, j + 1);
            }
        }
    }
}

```

4. ShakerSortableIntVector.cpp

```cpp
#include "ShakerSortableIntVector.h"

ShakerSortableIntVector::ShakerSortableIntVector(const int aArrayOfIntegers[], size_t aNumberOfElements)
    : SortableIntVector(aArrayOfIntegers, aNumberOfElements) {}

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

    while (l < r) {
        for (size_t i = l; i < r; i++) {
            if (!aOrderFunction(get(i), get(i + 1))) {
                swap(i, i + 1);
            }
        }
        r--;
        for (size_t i = r; i > l; i--) {
            if (!aOrderFunction(get(i - 1), get(i))) {
                swap(i - 1, i);
            }
        }
        l++;
    }
}

```